PETER NORTON

PROGRAMMIERHANDBUCH FÜR DEN IBM® PC

Das vollständige und umfassende Nachschlagewerk für die IBM Personal Computer

MICROSOFT® PRESS

VIEWEG

PETER NORTON

PROGRAMMIERHANDBUCH FÜR DEN IBM® PC

PETER NORTON

PROGRAMMIERHANDBUCH FÜR DEN IBM® PC

Das vollständige und umfassende Nachschlagewerk für die IBM Personal Computer

VIEWEG

Dieses Buch ist die deutsche Übersetzung von
The Peter Norton
Programmer's Guide to the IBM PC
The ultimate reference guide to the *entire* family of IBM®
personal computers
Microsoft Press, Bellevue, Washington 98009
Copyright © 1985 by Peter Norton
Übersetzung aus dem Amerikanischen:
Andreas Dripke und Angelika Schätzel, Wiesbaden

Framework™ ist ein Warenzeichen von Ashton-Tate. UNIX™ ist ein Warenzeichen der AT & T Bell Laboratories. COMPAQ® ist ein eingetragenes Warenzeichen, COMPAQ PLUS™ und DESK PRO™ sind Warenzeichen der COMPAQ Computer Corporation. CP/M® ist ein eingetragenes Warenzeichen der Digital Research Incorporated. Intel® ist ein eingetragenes Warenzeichen der Intel Corporation. IBM® ist ein eingetragenes Warenzeichen und PC-AT™, PC-DOS™, PC jr™, PC-XT™ und Topview™ sind Warenzeichen der International Business Machines Corporation. Microsoft® und XENIX® sind eingetragene Warenzeichen und GW-BASIC ist ein Warenzeichen der Microsoft Corporation. Motorola® ist ein eingetragenes Warenzeichen der Motorola Incorporated. Norton Utilities™ und Time Mart™ sind Warenzeichen von Peter Norton. Tandy® ist ein eingetragenes Warenzeichen von Radio Shack, einer Abteilung der Tandy Corporation. Prokey® ist ein eingetragenes Warenzeichen von Rose Soft. TJ® ist ein eingetragenes Warenzeichen und TJ Professional ist ein Warenzeichen von Texas Instruments.

Das in diesem Buch enthaltene Programm-Material ist mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Der Autor, die Übersetzer und der Verlag übernehmen infolgedessen keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programm-Materials oder Teilen davon entsteht.

1986

Alle Rechte vorbehalten
© Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig 1986

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlags unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Buchbinderische Verarbeitung: W. Langelüddecke, Braunschweig

ISBN 978-3-528-04489-3 ISBN 978-3-322-99366-3 (eBook)
DOI 10.1007/978-3-322-99366-3

Inhaltsverzeichnis

Vorwort VI
Dank VIII
1 Aufbau des PC 1
2 Interne Kommunikation 19
3 ROM-Software 41
4 Der Bildschirm 65
5 Grundlegendes über Disketten und Festplatten 95
6 Die Tastatur 123
7 Tonerzeugung 137
8 Grundlegendes über das ROM-BIOS 145
9 Bildschirmroutinen im ROM-BIOS 155
10 Disketten- und Plattenroutinen im ROM-BIOS 171
11 Die Tastaturroutinen im ROM-BIOS 187
12 Verschiedene BIOS-Routinen 193
13 Zusammenfassung: BIOS-Routinen 207
14 DOS-Grundlagen 221
15 DOS-Interrupts 227
16 Traditionelle DOS-Funktionen 247
17 Neue DOS-Funktionen 277
18 Zusammenfassung: DOS-Routinen 309
19 Erstellen eines Programmes 319
20 Programmiersprachen 329
Anhang A: Installierbare Schnittstellentreiber 361
Anhang B: Hexadezimale Arithmetik 367
Anhang C: Zeichen 377
Stichwortverzeichnis 389

Vorwort

Als ich dieses Buch schrieb, hatte ich ein einfaches, aber hochgestecktes Ziel vor Augen: Ihnen die Grundlagen der Programmierung der wichtigen IBM PC-Modelle zu vermitteln. Als IBM im Herbst 1981 ihren ersten Personal Computer vorstellte, war klar, daß das Gerät eine herausragende Bedeutung auf dem Mikrocomputersektor spielen würde. Im Laufe der Zeit stellte IBM eine Reihe neuer PCs vor, mittlerweile ist eine komplette Produktpalette daraus geworden. Der IBM PC gilt heute als der Industriestandard bei kommerziellen Mikrocomputern.

Um den IBM PC herum entstand eine eigene Branche, die Software für das Gerät liefert. Mit dem Aufkommen immer neuer PC-Modelle wurde allerdings die Entwicklung von Software ein zunehmend komplexer werdender Prozeß, da die Unterschiede der einzelnen Modelle zu berücksichtigen waren.
Ich möchte Ihnen das Grundlagenwissen zur Programmierung der IBM PC-Familie vermitteln und die wichtigsten Programmiertechniken aufzeigen. Dabei geht es nicht nur um die Auflistung technischer Details, sondern auch um die Erklärung der Konzepte. Im Vordergrund des Buches steht das Bestreben, alle wichtigen Mitglieder der PC-Modellreihe zu erfassen. Ich vertrete die Auffassung, daß man sich bemühen sollte, Programme so zu schreiben, daß sie auf allen Geräten lauffähig sind. Das erlaubt nicht nur die problemlose Portierung der Software von einem Modell auf ein anderes, sondern erhöht auch die Wahrscheinlichkeit, daß die Programme auf PCs laufen, die IBM in Zukunft vorstellen wird.

Als Zielgruppe sehe ich alle, die sich mit der Softwareseite des IBM PCs beschäftigen. Das sind in erster Linie Programmierer, aber auch beispielsweise Softwaremanager oder Anwender, die „hinter die Kulissen" sehen möchten.

Konzeptionen

Ich habe besonderen Wert darauf gelegt, bei allen technischen Informationen auch stets die dahintersteckenden Ideen und Konzepte zu erklären. Meiner Meinung nach hilft Ihnen das bei der Programmierung wesentlich mehr als ein reines Tabellenwerk. Wenn Sie die grundlegenden Aspekte kennen, können Sie sich leichter auch mit den Details auseinandersetzen.

Wie Sie von diesem Buch am meisten profitieren

Sie halten mit diesem Buch ein Werk in der Hand, daß als Lesebuch und als Nachschlagewerk gleichermaßen geeignet ist. Wenn Sie sich ausführlich mit der IBM PC-Familie beschäftigen möchten, werden Sie das Buch vermutlich Seite für Seite durchlesen. An den für Sie besonders interessanten Stellen machen Sie sich vielleicht Notizen oder streichen sie an, unwichtigere Abschnitte überfliegen Sie nur oder lassen sie aus. Falls Sie sich die Zeit für die ausführliche Lektüre nicht nehmen wollen, verwenden Sie das Buch als Nachschlagewerk. Da sich viele Sachverhalte nicht voneinander trennen lassen, finden Sie immer wieder Verweise, die Ihnen helfen, die Zusammenhänge zu verstehen. Um die Verweisstruktur nicht ausufern zu lassen, werden einige Sachverhalte an mehreren Stellen – meist in verschiedenen Zusammenhängen – angesprochen.

Ergänzende Literatur

Sie werden verstehen, daß man beim besten Willen nicht alle Informationen, die für die Programmierung des IBM PC von Bedeutung sind, in einem einzigen Buch zusammenfassen kann. Für Softwarespezialisten mag es beispielsweise notwendig sein, die PC-Hardware im einzelnen zu kennen. Für die meisten Programmierer genügt aber ein Grundlagenwissen über die Hardware, Schaltpläne sind für sie nicht interessant. Aus diesem Grund habe ich in das Buch nur einige ausgewählte Hardwareaspekte aufgenommen und die Ebene der Elektronik fast komplett weggelassen. Detaillierte technische Spezifikationen finden Sie in den *Technischen Handbüchern* der IBM.
Es kann ebenfalls nicht die Aufgabe des Buches sein, Sie mit den Grundlagen einer bestimmten Programmiersprache vertraut zu machen. Dafür gibt es Bücher über Assembler, BASIC, C, Pascal usw. Ich habe aber einige spezielle Aspekte der Programmiersprachen ausgesucht, die im Zusammenhang mit der PC-Programmierung von besonderem Interesse sind.
Es wird vorausgesetzt, daß Sie mit den Grundoperationen des Computers vertraut sind. Dazu gehört, daß Sie das Gerät einschalten und DOS laden und bedienen können. Falls Sie im Umgang mit DOS noch nicht ganz sicher sein sollten, nehmen Sie das DOS-Handbuch von IBM oder das Buch von Van Wolverton, MS-DOS, Verlag Vieweg, zu Hilfe.

Folgende andere Titel aus dem Verlag Vieweg mögen für Sie ebenfalls von Interesse sein:

MS-DOS (Versions 1.0–3.2) Technical Reference Enzyclopedia, Microsoft Reference Library hrsg. von Microsoft Corp. USA.

Damit genug der Vorrede, wenden wir uns unserem Thema – der Programmierung des IBM PC – zu.

Dank

So viele haben zu der Erstellung des Buches beigetragen, daß es unmöglich ist, sie alle hier zu nennen. Stellvertretend für alle anderen möchte ich besonders Suzanne Ropiequet für ihren Einsatz danken.

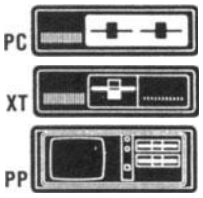
Peter Norton

Kapitel 1
Aufbau des PC

1.1 8088 Mikroprozessor 2
1.1.1 80286 Mikroprozessor 5
1.1.2 8087 Arithmetik-Coprozessor 7
1.2 Support-Chips 8
1.2.1 8259 Interrupt-Controller 9
1.2.2 8237A DMA-Controller 9
1.2.3 8284A Taktgenerator 10
1.2.4 8255 Programmierbares Peripherie-Interface 10
1.2.5 8253 Programmierbarer Zeitgeber 10
1.2.6 6845 Bildschirm-Controller 11
1.2.7 PD765 Disketten-Controller 11
1.3 Der Bus 11
1.3.1 Adreßbus 12
1.3.2 Datenbus 12
1.4 Speicherbausteine 13
1.5 Konzeption des PC 16

Aus der Sicht des Programmierers bestehen alle PC-Modelle aus folgenden Grundelementen: Prozessor, Speicherbausteinen und mehreren intelligenten oder programmierbaren Chips für diverse Aufgaben. Die wichtigsten Bauteile, die der Computer zum Arbeiten benötigt, sind auf der System- oder Hauptplatine untergebracht. Andere, ebenfalls wichtige Teile befinden sich auf Erweiterungskarten, sogenannten *Add-on-Boards*. Diese können in entsprechende Erweiterungsplätze der Hauptplatine eingesteckt werden.
Die Systemplatine beherbergt einen Mikroprozessor, entweder den 8088 oder den 80286, der einen Speicher von mindestens 64 Kbyte benötigt. Außerdem befinden sich auf der Platine ROM-Speicherchips mit ROM-BIOS und BASIC und verschiedene wichtige Support-Chips. Einige dieser ICs (*Integrated Circuit* oder IC ist gleichbedeutend mit *Chip* oder auch *Baustein*) kontrollieren externe Geräte, beispielsweise die Diskettenstation oder den Bildschirm, andere helfen dem Mikroprozessor, seine Aufgaben zu erfüllen.
Im vorliegenden Abschnitt wollen wir die wichtigsten technischen Aspekte aller relevanten ICs durchsprechen. Oft existieren für ein und denselben Chip unterschiedliche Bezeichnungen in der Literatur, einige davon werden im weiteren Verlauf erwähnt. Dazu an dieser Stelle ein kleines Beispiel: Teile der Peripherie des PC, beispielsweise die Tastatur, unterstehen einem Chip, der als 8255 oder genauer 8255A-5 bekannt ist. Weitaus häufiger wird der Chip aber auch nach seiner Funktion benannt: *Programmable Peripheral Interface* (Programmierbares Peripherie-Interface) oder kurz PPI. Alle diese Namen beziehen sich auf ein und denselben Baustein.

1.1 8088 Mikroprozessor



Der 8088 ist ein 16-bit-Prozessor, der das Herz des Standard IBM PC darstellt. Der Portable, der XT und der Original-PC enthalten den 8088. Er ist die Zentraleinheit oder CPU (*Central Processing Unit*). Die CPU ist das Gehirn der Maschine, fast jedes Bit, das in irgendeiner Form verarbeitet oder transferiert wird, passiert sie.
Der 8088 kontrolliert die Basisoperationen des Computers durch Aussenden und Empfangen von Kontrollsignalen, Speicheradressen und Daten, die über den sogenannten *Bus*, ein Netzwerk elektronischer Wege, der die einzelnen Teile des Computers miteinander verbindet, laufen. Entlang des Busses sind Eingabe-(Input)- und Ausgabe-(Output)-Ports (E/A oder I/O) vorhanden, die die verschiedenen Speicher- und Support-Chips miteinander verbinden. Daten durchlaufen die E/A-Ports, wenn sie von der CPU zu anderen Teilen des Computers oder von diesen zur CPU geschickt werden.

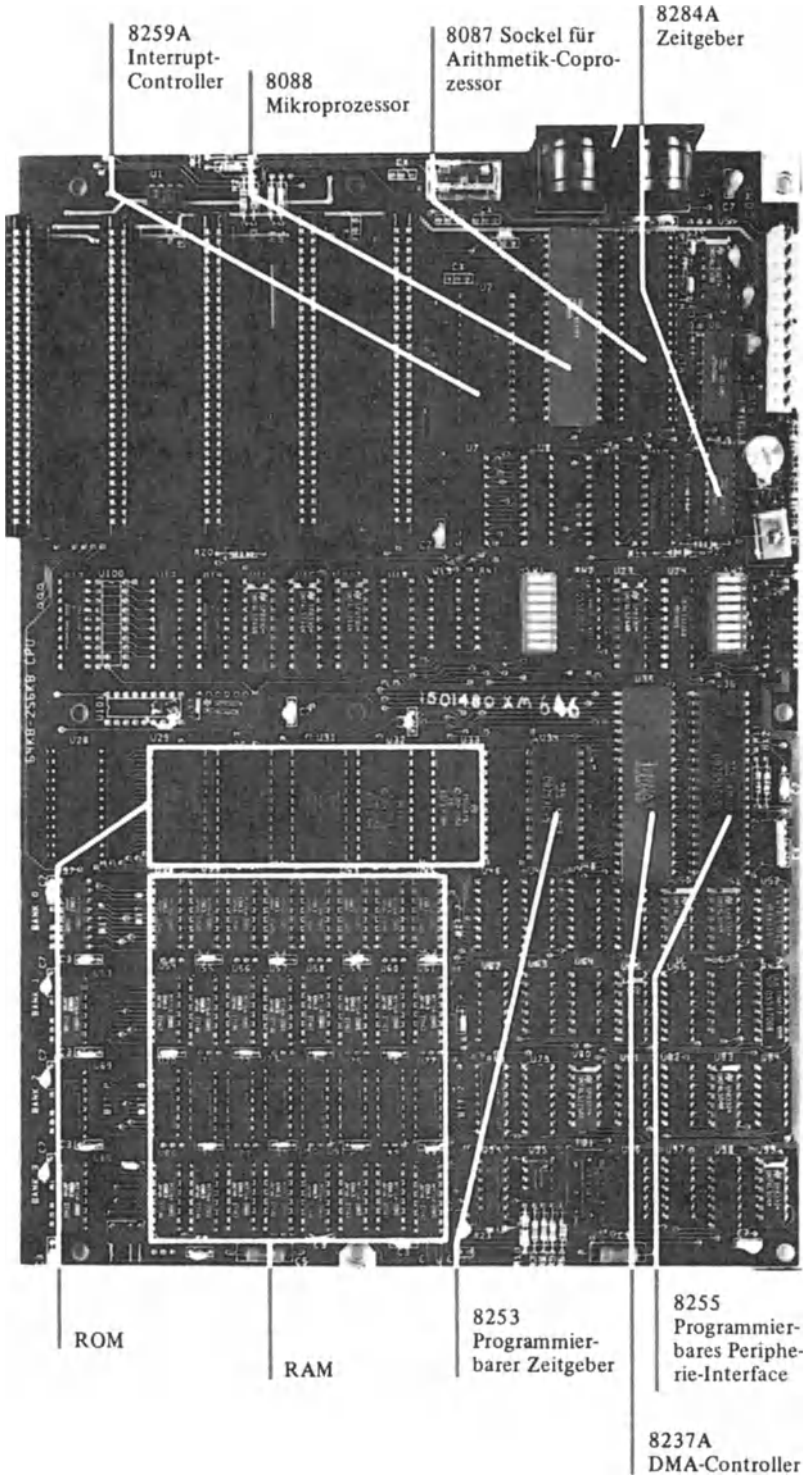


Bild 1-1 Die Systemplatine des PC/XT

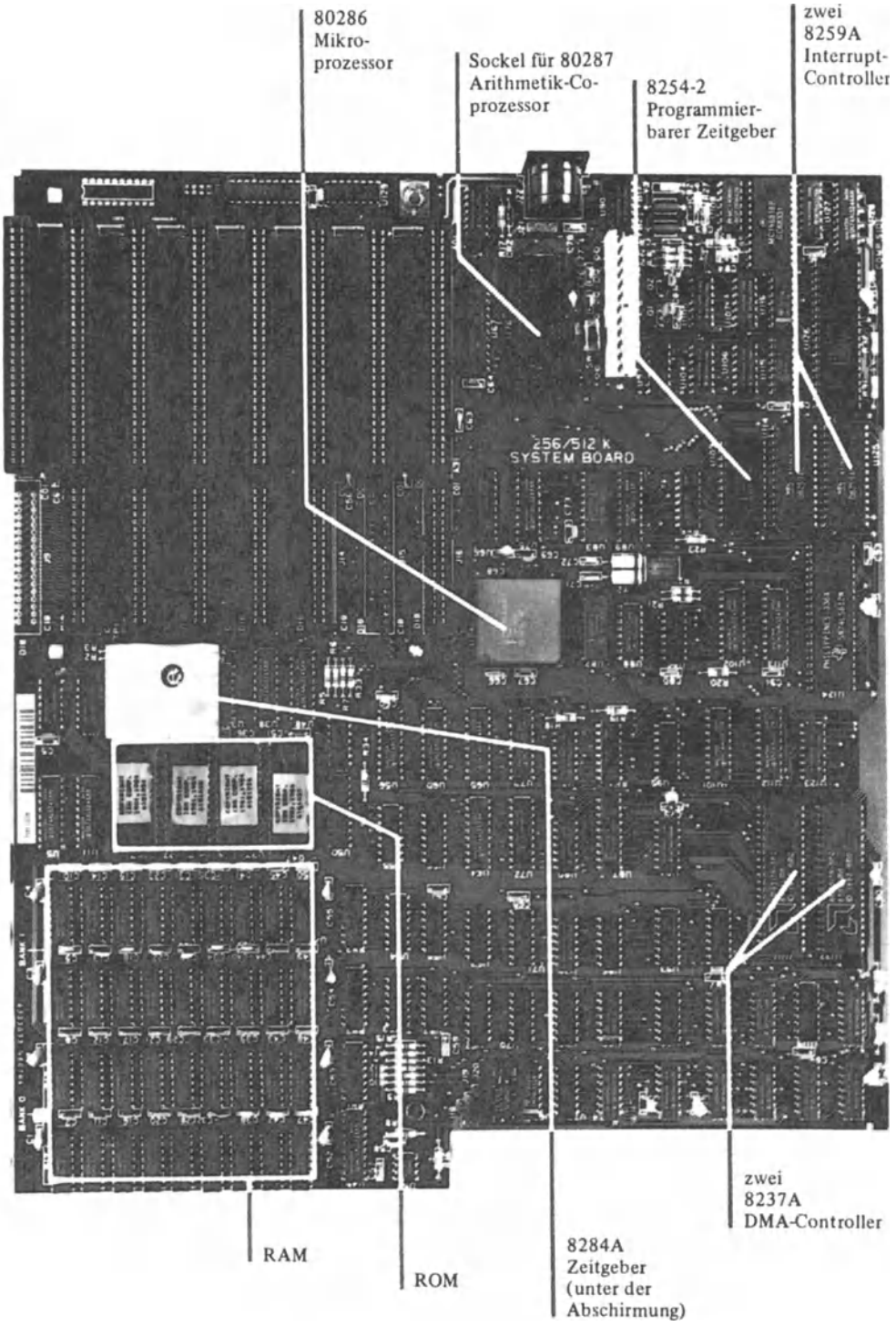


Bild 1-2 Die Systemplatine des AT

Innerhalb des 8088 sind 14 Register als Arbeitsbereich für den Datentransfer und die Verarbeitung vorhanden. Diese internen Register mit einem Gesamtumfang von 28 Bytes können zeitweise Daten, Speicheradressen, Befehlszeiger und Status- und Kontrollflaggen speichern. Statt von *Flaggen* wird manchmal auch von *Merkern* oder *Kennzeichen* geredet. Wir wollen in diesem Buch aber *Flagge* sagen, weil das dem englischen Begriff *Flag* am nächsten kommt. Durch die Register ist der 8088 in der Lage, auf bis zu 65.536 E/A-Ports und über eine Million Speicherstellen zuzugreifen. In Kapitel 2 finden Sie nähere Hinweise über die Arbeitsweise des 8088.

Der "Stammbaum" der 8088-Familie: Der 8088 ist nur ein Mitglied einer "eng verwandten Familie" von 16-bit-Mikroprozessoren, die allesamt von dem amerikanischen Halbleiterspezialisten Intel hergestellt werden. Das Grundmodell ist der 8086. Der 8088 und der 8086 unterscheiden sich nur in einem einzigen kleinen, aber nicht unwichtigem Aspekt: Obwohl der 8088 ein 16-bit-Prozessor ist, benutzt er einen 8-bit-Bus im Gegensatz zum 16-bit-Bus des 8086. Der Unterschied zwischen einem 8-bit-Bus und einem 16-bit-Bus wird in Kapitel 1.3.2 behandelt. Tatsächlich trifft alles, was Sie über den 8086 lesen, auch auf den 8088 zu. In Bezug auf die Programmierung verhalten sich beide völlig identisch.
Wie bereits gesagt, ist der 8088 nur einer der Intel 16-bit-Prozessoren, obgleich lange Zeit der wichtigste in IBMs PC-Reihe. Andere Intel CPUs werden häufig in kompatiblen Rechnern benutzt. Der schon erwähnte 8086 ist beispielsweise das Gehirn des Compaq Deskpro, eines PC-kompatiblen Computers. Der 80188 und der 80186 (kurz: 188 und 186), die die verbesserten Ausführungen der Originale 8088 und 8086 sind, kommen in einer Vielzahl anderer Mikrocomputer, die dem IBM PC zum Teil recht ähnlich sind, zum Einsatz. Diese zwei Prozessoren besitzen insgesamt eine größere Leistungsfähigkeit als ihre Vorgänger, ihr Hauptvorteil liegt jedoch darin, daß sie die eigentliche Zentraleinheit und wichtige und notwendige Support-Operationen in sich vereinen. Diese Support-Operationen werden beim 8088 und beim 8086 extern durchgeführt. Ungeachtet der vielfältigen Verbesserungen sind der 186 und der 188 noch lange nicht der letzte Stand der Technik, soweit es die PC-Modellreihe betrifft.

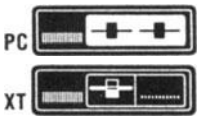
1.1.1 80286 Mikroprozessor

Der am weitesten entwickelte Mikroprozessor von Intel, der derzeit in IBMs PC-Produkten Verwendung findet, ist der 80286 (oder kurz: 286). Dieser Baustein bildet das Herz des IBM AT. Der 80286 ist ein echter 16-bit-Prozessor, der auch einen 16-bit-Bus vorweisen kann und eine ganze Reihe neuer Programmiereigenschaften gegenüber seinen Vorgängern aufweist. Vielleicht sind die wichtigsten neuen Leistungsmerkmale des 286

seine Fähigkeit zu Multitasking und virtueller Speicherverwaltung - zwei Konzepte, die jedem, der sich mit Großrechnern auskennt, geläufig sind.
Multitasking ist die Fähigkeit der CPU, zur gleichen Zeit verschiedene Programme zu bearbeiten. Gleichzeitiges Ausdrucken eines Textes und Berechnen einer Kalkulationstabelle wäre ein Beispiel dafür. Das wird ermöglicht, indem die CPU ihre Aufmerksamkeit schnell zwischen den einzelnen Programmen hin- und herspringen läßt. Ein normaler PC kann in begrenztem Umfang auch Mutitasking ausführen, er benötigt dafür allerdings hochentwickelte Software, wie IBMs "Topview" oder Microsofts "Windows". Ein echter Multitasking-Prozessor erledigt diese Aufgabe intern mit Hilfe des Betriebsprogrammes. Da die Multitasking-Fähigkeit des 286 also hauptsächlich auf der Hardware basiert, ist dieses Konzept wesentlich schneller und zuverlässiger als software-getriebenes Multitasking.
Virtuelles Speichern erlaubt dem Computer, sich zu verhalten, als hätte er mehr Hauptspeicherkapazität zur Verfügung, als dies in Wirklichkeit der Fall ist. Durch ein extrem hochentwickeltes Software- und Hardware-Design ist es möglich, einem Programm eine Speicherkapazität von bis zu einem Gigabyte (eine Million Byte) zur Verfügung zu stellen, obwohl die Hardwarespeicherbausteine tatsächlich vielleicht nur einen Bruchteil dieser Größe ausmachen. Die Täuschung ergibt sich durch ein sorgfältig ausgearbeitetes Speicheradressierungskonzept, bei dem nur Teile des Programmes im Hauptspeicher ablegt, andere aber auf Diskette oder Festplatte gespeichert werden. Falls bestimmte Befehle oder Programmdaten benötigt werden, die sich gerade nicht im Hauptspeicher befinden, werden sie von der Diskette/Platte nachgeladen. Der 286 und das Betriebssystem können erkennen, wo sich die jeweils benötigten Informationen befinden, und wohin sie gebracht werden müssen, damit das Programm reibungslos und effizient arbeiten kann - trotz der Verteilung auf Hauptspeicher und externen Speicher (Diskette bzw. Festplatte).
Bei Mini- und Großcomputern ist virtuelles Speichern seit langem üblich, während es im Mikrocomputerbereich erst seit relativ kurzer Zeit gebräuchlich ist. Die Einführung dieses Konzeptes mit dem 286 im IBM PC AT hat sicherlich tiefgreifende Auswirkungen auf Anwendersoftware, da es nunmehr möglich ist, benutzerfreundliche Programme zu schreiben, die in ihrer Länge nur durch die physikalische Speicherkapazität der Diskette bzw. Platte begrenzt werden.
Der AT wird oft als größeres, leistungsfähigeres Modell der PC-Reihe angesehen, das in der Lage ist, die gängige PC-Software, sowohl das DOS-Betriebssystem als auch die meisten Anwendungsprogramme, zu verarbeiten. Das stimmt aber nur bedingt, nämlich nur dann, wenn man Multitasking und das virtuelle Speicherkonzept aus der Betrachtung herausnimmt. Verwendet man diese Eigenschaften, wird der AT im Grunde zu einem anderen Computer, der ein anderes Betriebssystem und andere Software erfordert. Das macht den AT zum Mitglied einer völlig neuen Generation von Mikrocomputern, die sich von der ursprünglichen

PC-Konzeption abgespalten hat. Die meisten in diesem Buch behandelten Programmiertechniken beziehen sich auf den 8088. Sollte es sich als notwendig erweisen, wird auch auf den 8086 eingegangen.
Alle Mitglieder der 8088/8086-Familie wurden so entworfen, daß sie Hilfsprozessoren unterstützen. Sie arbeiten auch mit zwei speziellen Coprozessoren: dem 8087 Arithmetik-Coprozessor und dem 8089 I/O-Coprozessor. Die zusätzlichen Bausteine dienen der Entlastung der Haupt-CPU in bestimmten Arbeitsbereichen. Der IBM PC bietet nur Platz für den 8087, so daß wir diesen Coprozessor im folgenden Abschnitt genauer ansehen wollen.

1.1.2 8087 Arithmetik-Coprozessor



Der 8088 kann nur mit ganzen Zahlen arbeiten. Für Real- oder Fließkommazahlen sind spezielle Abläufe nötig. Diese werden gewöhnlich von Unterprogrammen übernommen, die die Bearbeitung von Fließkommazahlen wirkungsvoll, aber mit einem großen Verlust an Zeit und Leistungsfähigkeit bewerkstelligen.
Der 8087 Arithmetik-Coprozessor bearbeitet Fließkommazahlen 10 bis 50 mal schneller, als es dem 8088 möglich ist. Außerdem ist die Rechengenauigkeit wesentlich höher als beim 8088 (übrigens auch gegenüber den meisten mehrere Millionen DM kostenden Großrechnern). Außer den einfachen arithmetischen Funktionen (Addieren, Subtrahieren, Multiplizieren und Dividieren) besitzt der 8087 eingebaute trigonometrische Funktionen (Sinus, Cosinus, Tangens usw.), die komplexes Programmieren wesentlich vereinfachen. Weiterhin kann er Zahlen verschiedenen Formats bearbeiten: ganze Zahlen, Fließkommazahlen und dezimale Formate sind möglich. Der 8087 führt diese Funktionen aus, während der 8088 gleichzeitig andere Aufgaben erfüllt.

Jedes PC-Modell, das auf dem 8088 basiert, kann mit dem 8087 arbeiten, wobei allerdings eine spezielle Softwareunterstützung nötig ist. Der AT benutzt den 80287 Arithmetik-Coprozessor, der eine neuere Version des 8087 darstellt und auf den 80286 abgestimmt ist. Obwohl der 8087 nahezu alle arithmetischen Funktionen des IBM PC übernehmen kann, macht nur relativ wenig Software Gebrauch davon. Diese unglückliche Situation beruht auf der einfachen historischen Gegebenheit, daß IBM den 8087 nicht von Anfang an unterstützte, obwohl er schon im PC vorhanden war.
Genau wie in den meisten anderen Modellen, unterstützte IBM den 8087 solange nicht, bis die Standards der PC-Hardware und Software auf dem Markt etabliert waren. Das hatte zur Folge, daß ein Großteil der für den IBM PC entwickelten Software und Hardware den 8087 nicht berücksichtigt.
Trotz der so lange eingeschränkten Verbreitung des 8087-Chips und der dazugehörenden Software vermehren sich nun aber die auf dem Markt erhältlichen Programme, wie beispielsweise Ashton-Tates "Framework", die

Typ	Ungefährer Wertebereich			Signifikante Stellen Bits	(dezimal)
Integerwort	– 32.768	–	+ 32.767	16	4
kurze Integerzahl	– 2 X 10E9	–	+ 2 X 10E9	32	9
lange Integerzahl	– 9 X 10E18	–	+ 9 X 10E18	64	18
Dezimal gepackt	– 99 ... 99	–	+ 99 ... 99	80	18
kurze Realzahl	8,43 X 10E – 37	–	3,37 X 10E38	32	6–7
lange Realzahl	4,19 X 10E – 307	–	1,67 X 10E308	64	15–16
temporäre Realzahl	3,4 X 10E – 4932	–	1,2 X 10E4932	80	19

Tabelle 1-1 Der Wertebereich der numerischen Datentypen, die mit den acht 80-bit-Registern des 8087 oder 80287 verarbeitet werden können.

nicht nur die Vorteile des 8087 ausnutzen, sondern selbsttätig erkennen, ob der 8087 eingesetzt ist und je nachdem den Arithmetik-Coprozessor ansteuern oder umgehen. Es bleibt zu hoffen, daß sich zukünftig immer mehr Software dieser Technik bedient.
Da der Einsatz des 8087 bis heute nur relativ selten anzutreffen ist, behandeln wir die damit verbundenen speziellen Programmierprobleme in diesem Buch nicht.

1.2 Support-Chips

Der Mikroprozessor kann nicht den gesamten Computer ohne fremde Hilfe überwachen, er soll es auch nicht. Indem bestimmte Arbeiten an andere Bausteine delegiert werden, kann sich die CPU ihren Hauptaufgaben widmen. Support-Chips können z.B. den Informationsfluß durch interne Bausteine übernehmen. Das ist die Funktion des Interrupt- und des DMA-Controllers. Auch der Datenfluß von und zu bestimmten Geräten (Diskettenstation oder Bildschirm) wird von Support-Chips kontrolliert. Diese Geräte- oder Schnittstellen-Controller sind oft auf separaten Karten untergebracht, die in die Erweiterungsschächte des PC eingesteckt werden können.
Viele Support-ICs sind programmierbar, das heißt, sie können auf spezielle Aufgaben zugeschnitten werden. Vom direkten Programmieren dieser Bausteine ist in den meisten Fällen allerdings dringend abzuraten. In den folgenden Abschnitten wird jeweils darauf hingewiesen, ob die Programmierung des angesprochenen Bausteins mit den Funktionen des Computers kollidieren kann oder nicht. Da dieses Buch nicht die direkte Hardwarekontrolle zum Thema hat, sei zum Programmieren der einzelnen ICs auf das *Technische Referenzhandbuch* von IBM verwiesen.

1.2.1 8259 Interrupt-Controller

Der 8259 überwacht die sogenannten *Interrupt-Operationen*. Interrupts (zu deutsch: Unterbrechungen) sind Signale, die von der Peripherie zur CPU geschickt werden, um deren Aufmerksamkeit zu erwecken oder mitzuteilen, daß eine bestimmte Handlung erfolgt ist. Der 8259 empfängt die Interruptsignale und vergleicht deren Priorität. Je nach Beurteilung wird der Interrupt an die CPU gemeldet. Sobald diese einen Interrupt erhält, ruft sie ein bestimmtes, dem Peripheriegerät zugeordnetes, Programm auf. Dieses erledigt die erwartete Reaktion. In Kapitel 2 und 3 werden Interrupts ausführlicher behandelt.

Der 8259 vermag bis zu acht Interruptsignale gleichzeitig zu bearbeiten. Für eine höhere Bearbeitungskapazität können zwei 8259 miteinander gekoppelt werden. IBM hat sich beim AT dieser Möglichkeit bedient und tatsächlich zwei 8259 zusammengeschlossen, so daß bis zu 15 Interrupts gleichzeitig ausgeführt werden können.

Im allgemeinen wird der 8259 nicht programmiert, da jede Veränderung leicht mit den Basisoperationen des Computers in Konflikt geraten könnte. Grundsätzlich ist es jedoch möglich, die Prioritätsbedingungen während der Ausführung eines Programmes jederzeit zu rekonfigurieren. Das bedeutet, das Programm kann die Reihenfolge der Interrupt-Abarbeitung durch den 8259 ändern, um den Wünschen des Programmierers gerecht zu werden.

Andere Bezeichnungen für den 8259 sind 8259A, INTR und PIC (*Programmable Interrupt Controller*).

1.2.2 8237A DMA-Controller

Um den Mikroprozessor nicht ständig mit Datentransfer zu belasten, ist es einigen Komponenten des Computers möglich, Daten vom und zum Speicher zu übertragen, ohne die CPU zu belasten. Die Operation wird direkter Speicherzugriff oder DMA (*Direct Memory Access*) genannt und unterliegt dem Baustein 8237A oder dem DMA-Controller. Die Hauptaufgabe dieses ICs besteht darin, der Diskettenstation das Lesen und Schreiben von Daten zu ermöglichen, ohne daß die Daten die CPU passieren müssen. Da Diskettenein-/ausgabe eine relativ langsame Angelegenheit ist, beschleunigt DMA die Abarbeitung eines Programmes merklich, da die CPU zeitlich entlastet wird. Alle IBM PCs arbeiten mit dem DMA-Baustein 8237A oder einem vergleichbaren IC.

Der DMA-Controller hat vier verschiedene Kanäle, um Daten vom und zum Speicher zu transferieren und einen internen Zwischenspeicher mit 344 Bits. Es ist möglich, mehrere DMA-Controller miteinander zu koppeln, beim AT wird davon Gebrauch gemacht.

1.2.3 8284A Taktgenerator

Der Taktgenerator erzeugt mehrphasige Taktsignale, die zum Betreiben des Mikroprozessors und der Peripherie nötig sind. Seine Grundfrequenz liegt bei 14,3228 Megahertz (MHz oder Millionen Schwingungen pro Sekunde). Die anderen ICs dividieren diese Grundfrequenz durch eine Konstante, um die für sie jeweils nötigen Taktfrequenzen zu erhalten. Der im Standard-PC vorhandene 8088 wird mit 4,77 MHz betrieben, einem Drittel der Grundfrequenz. Die neuen Versionen des Mikroprozessors 8086 sind schneller. Der 8088-2, der in einigen PC-Modellen Verwendung findet, arbeitet mit einer Frequenz von 8 MHz. Das entspricht nahezu einer Verdoppelung der Geschwindigkeit des 8088. Der 80286 arbeitet mit 6 MHz und ist damit 1,5 mal schneller als der 8088. Der interne Bus und der 8253 Programmierbare Zeitgeber arbeiten mit einer Frequenz von 1,193 MHz, laufen also mit einem Viertel der 8088-Taktfrequenz oder einem Zwölftel der Grundfrequenz.

1.2.4 8255 Programmierbares Peripherie-Interface

Mit Hilfe des 8255 werden spezielle Peripheriegeräte mit der CPU verbunden. Daten, die zu oder von diesen Geräten, z.B. Lautsprecher oder Kassettenlaufwerk, transferiert werden, passieren die Ein-/Ausgabeports unter der Kontrolle dieses Bausteines.
Der 8255 ist auch unter den Namen 8255A-5 oder PPI (*Programmable Peripheral Interface*) bekannt. Er wird normalerweise von der Systemsoftware angesprochen und braucht deshalb nicht programmiert zu werden, obgleich das durchaus möglich ist.

1.2.5 8253 Programmierbarer Zeitgeber

Der 8253 ist ein Vielzweck-Zeitgeber und Zähler, der bis zu drei exakten Zeitverzögerungen unter Softwarekontrolle generieren kann. Er erhält sein Signal vom 8259A Taktgenerator und arbeitet selbst mit einer Frequenz von 1,190 MHz.
Der 8253 wird hauptsächlich zur Tonerzeugung auf dem internen Lautsprecher benutzt. Aber auch andere frequenzabhängige Funktionen, wie Diskettenein- und -ausgabe und Zeitmessung, werden von diesem Baustein unterstützt. In Kapitel 7 finden Sie Details zur Tonerzeugung.
Häufig wird der 8253 einfach *Zeitgeber* oder *Timer* genannt, manche sprechen vom 8253-5.

1.2.6 6845 Bildschirm-Controller

Der 6845 Bildschirm-Controller oder auch CRT-Controller (*Cathode Ray Tube*) ist auf einer Erweiterungsplatine untergebracht, die *Bildschirmadapter* heißt. Er besitzt 19 interne Register, die zum Definieren und Kontrollieren eines Rasterbildschirmes gebraucht werden. Dieses IC ist zwar programmierbar, aber auch hier sei Ihnen wiederum der Rat gegeben, die Kontrolle dem BIOS zu überlassen. Mehr über den Bildschirmadapter finden Sie in Kapitel 4.

1.2.7 PD765 Disketten-Controller

Der PD765 ist für einen reibungslosen Ablauf der Diskettenoperationen verantwortlich. Weitere gängige Namen sind FDC (*Floppy Disk Controller*) und manchmal auch NEC (nach der Herstellerfirma NEC "Nippon Electric Company"). Wie den schon behandelten 6845 Bildschirm-Controller sollten Sie auch diesen Baustein der BIOS-Kontrolle überlassen.

1.3 Der Bus

Alle internen Kontrolleinheiten stehen über den Bus miteinander in Verbindung. Der Bus ist ein zentraler Teil der Hauptplatine, an den praktisch alle Einheiten angeschlossen werden. Beim Datentransfer passieren alle Daten diesen gemeinsamen Weg, um ihr Ziel zu erreichen.
Alle Kontrollbausteine und Speicherzellen sind direkt oder indirekt mit dem Bus verbunden. Werden neue Komponenten in die Erweiterungsplätze des Computers eingesteckt, entspricht das einer direkten Verbindung mit dem Bus. Der neue Zusatz ist nun ein allen anderen Komponenten gleichwertiger Partner. Sämtliche Daten, die den Computer durchlaufen, werden mindestens in einer von mehreren, dem Bus angehörenden Speicherstellen zwischengelagert. Die meiste Zeit sind die Daten im Hauptspeicher abgelegt, der im PC aus vielen tausend 8-bit-Speicherzellen besteht. Manche Daten werden aber zu einem Port oder Register transferiert, wo sie verbleiben, bis die CPU sie zu ihren endgültigen Speicherplätzen weiterleitet. Die Ports und Register speichern im allgemeinen nur ein bis zwei Bytes zur gleichen Zeit und werden hauptsächlich als Puffer für auszutauschende Daten benutzt. Nähere Informationen über Ports und Register stehen in Kapitel 2.
Speicherzellen und Ports, die als Speicherplätze fungieren, erhalten eine Adresse, die allein die Position der Speicherstelle identifizieren kann. Sobald Daten zum Transfer bereit sind, wird zuerst die Zieladresse - das ist die Adresse der Speicherstelle, zu der die Daten gesendet werden sollen - übertragen. Erst dann kann der Datenfluß beginnen. Der Bus überträgt aber nicht nur einfache Daten, sondern auch Leistungs- und Kon-

trollsignale, wie Zeitsignale des Systemtakts oder Interruptsignale, und auch die Adressen von Tausenden von Speicherzellen und extern angeschlossenen Geräten. Um diese verschiedenen Funktionen in Einklang zu bringen, ist der Bus in vier Bereiche unterteilt: die Leistungsführungen, den Kontrollbus, den Adreßbus und den Datenbus. Sowohl der Adreßbus als auch der Datenbus werden hier eingehend behandelt, da dies wertvolle Informationen über wichtige Eigenschaften des Computers vermittelt.

1.3.1 Adreßbus

Der Adreßbus des Standard-PC enthält 20 Signalwege, um die Adressen von Speicherzellen und angeschlossenen Zusatzgeräten übertragen zu können (Adressenspeicher werden in den Kapiteln 1.4 und 3 näher behandelt). Es gibt zwei mögliche Werte (1 oder 0), die von den 20 Adreßleitungen transportiert werden können. Aufgrund dieser Tatsache sind die Standard-PCs in der Lage 2^{20} Adressen, das entspricht einer Zahl von über einer Million, zu unterscheiden. Der AT verfügt sogar über 24 Adreßleitungen, wodurch ihm eine Differenzierung von 2^{24} oder über 16 Millionen Adressen möglich ist.

AT

1.3.2 Datenbus

Der Datenbus steht mit dem Adreßbus in Verbindung, um Datenströme vom und zum Computer transferieren zu können. Der 8088-PC benutzt einen Datenbus, der über acht Signalleitungen verfügt. Jede dieser Verbindungen überträgt eine binäre Information (ein Bit). Die Informationen werden also in 8-bit- oder 1-byte-Einheiten durch den Bus geleitet. Der 80286 des AT arbeitet mit einem Datenbus mit 16 Signalwegen, so daß 16 Bits gleichzeitig übertragen werden können.

AT

Nun ist ja auch der 8088 ein 16-bit-Prozessor, der folglich 16 Bits auf einmal verarbeiten kann. Den Datenaustausch mit seiner Umgebung erledigt er allerdings in Einheiten von 8 Bit. Wir haben es also mit einem 16-bit-Prozessor mit 8-bit-Datenbus zu tun. Aus diesem Grunde wird der 8088 auch oftmals fälschlich als 8-bit-Prozessor bezeichnet. Richtig ist, daß zwar eine 16-bit-CPU vorliegt, die aber in ihrer Verarbeitungsgeschwindigkeit dem "echten" 16-bit-Prozessor 80286 (mit 16-bit-Datenbus) deutlich unterlegen ist. Der Unterschied in der Abarbeitungsgeschwindigkeit der beiden Prozessoren resultiert zum überwiegenden Teil allerdings nicht direkt aus der Datenübermittlungsgeschwindigkeit, sondern beruht auf der höheren Taktfrequenz und der verbesserten internen Organisation des 80286.

Für die Benutzung des 8088 in so vielen Computern, vor allem den älteren Modellen, gibt es eine gewichtige Ursache, die im wesentlichen wirtschaftlich begründet ist. Es ist nämlich noch nicht so lange her, daß 8-bit-ICs in sehr großen Stückzahlen und äußerst billig am Markt angeboten

wurden. Warum sollte man angesichts dieser Tatsache also zu den teuren 16-bit-Mikroprozessoren greifen, die zudem nur in kleineren Mengen auf dem Markt zur Verfügung standen. Der 8088 war aber zum Zeitpunkt der Entwicklung des IBM PC dem 8086 mit seinem 16-bit-Bus nicht nur aus Kostengründen vorzuziehen, sondern auch, um ein Mindestangebot an Zusatzbausteinen zur Verfügung zu haben. Das war bei der damaligen Knappheit an 16-bit-ICs nur bei diesem Prozessor möglich. Heute sind die Preise der 16-bit-Bausteine wesentlich gesunken, so daß der Verwendung des 80286 in einem Computer und des damit verbundenen 16-bit-Datenbusses nichts mehr im Wege steht. Der 80286 kann weiterhin jegliche Kombination aus 8-bit- und 16-bit-Einheiten verarbeiten, wodurch die Kompatibilität unter den einzelnen PC-Modellen erhalten bleibt.

1.4 Speicherbausteine

Wir haben nun die CPU, die Support-Bausteine und den Bus behandelt, wobei der Speicher immer nur gestreift wurde. Ganz bewußt setzen wir die Diskussion des Speichers an das Ende dieses Kapitels, weil Speicherbausteine im Gegensatz zu den vorher besprochenen Bausteinen den Datentransfer weder beeinflussen noch kontrollieren können. Ihre einzige Aufgabe besteht darin, Daten auf Abruf zu speichern.
Die Anzahl der Speicherchips, die im Computer existieren, legt die maximale Speichergröße für Programme und andere Daten fest. Die Kapazität variiert bei den unterschiedlichen Modellen. Der Standard-PC ist gewöhnlich mit einem ROM-Speicher (*Read Only Memory*, Festwertspeicher) von ungefähr 40 Kbyte ausgestattet, für Erweiterungen ist entsprechender Platz vorgesehen. Der Umfang des RAM-Speichers (*Random Access Memory*, Schreib-/Lesespeicher) beträgt normalerweise 128 bis über 512 Kbyte. Da auf der Hauptplatine des Original-PC maximal 256 Kbyte untergebracht werden können, gibt es die Möglichkeit, Speicherkarten mit unterschiedlicher Kapazität zusätzlich in den Erweiterungssteckplätzen zu installieren. Das ist aber nur der physikalische Aspekt der Speichererweiterung, für die CPU sind diese Speichereinheiten nichts weiter als einige Tausend 8-bit-(1-byte)-Speicherzellen mit entsprechend zugeordneten Adressen.
Auch Programmierer müssen sich dieser Denkweise bedienen. Wichtig ist nicht, wieviel physikalischer Speicherplatz zur Verfügung steht, sondern wieviele adressierbare Speicherstellen vorhanden sind. Die maximale Anzahl an Adressen, die der 8088 verwalten kann, beläuft sich auf 1024 Kbyte oder 1.048.576 Bytes; mehr Speicherstellen sind grundsätzlich nicht möglich. Die Speicheradressierung wird in Kapitel 2.2 näher erläutert.
Jedem Byte ist eine numerische 20-bit-Adresse zugeordnet. Das Speicherkonzept des 8088 legt 20-bit-Adressen fest, die über die 20 Signalleitungen des Adreßbusses geleitet werden müssen (der Adreßbus wird in

Kapitel 1.3.1 behandelt). Eine 20-bit-Adresse wird im allgemeinen als 5-stellige Hexadezimalzahl oder Hexzahl dargestellt, weil sie für den Menschen in dieser Form leichter zu lesen ist. Die Werte der Adressen liegen zwischen hex 00000 und hex FFFFF (dez. 0 bis 1.048.576). Sollte Ihnen der Begriff der Hexzahlen nicht geläufig sein, lesen Sie bitte in Anhang B nach.
Üblicherweise wird der 1024 Kbyte umfassende adressierbare Gesamtspeicherplatz in 16 Blöcke zu je 64 Kbyte Länge unterteilt. Diese 64-Kbyte-Blöcke werden an der ersten Stelle ihrer Hex-Adresse (der höchstwertigsten Stelle) identifiziert. Alle Speicherzellen innerhalb eines Blocks besitzen die gleiche Anfangsstelle und sind auf diese Weise eindeutig diesem bestimmten Block zugeordnet. Die ersten 64 Kbyte des Speichers gehören dem 0-Block an, der die Byte-Adressen hex 00000 bis hex 0FFFF umfaßt. Der letzte Block ist der F-Block mit den Adressen hex F0000 bis hex FFFFF.

Nun gibt es zwischen den einzelnen Blöcken nur wenige funktionale Grenzen, vielmehr geschieht die Einteilung in Blöcke einerseits aus Bequemlichkeit, andererseits, weil das Gesamtspeicherkonzept des PC, je nach Modell, den verschiedenen Blöcken unterschiedliche Aufgaben zuteilt.
Theoretisch kann in jedem Speicherbereich ROM oder alternativ RAM untergebracht sein. Es ist jedoch üblich, die ersten zehn Blöcke (Block 0 bis Block 9) mit einer Gesamtkapazität von 640 Kbyte für RAM freizuhalten und quasi als gewöhnlichen Speicherplatz zu benutzen. Jeder Speicher, der in einem PC enthalten ist, beginnt beim ersten Block. Da RAM

F 0 0 0	ROM-Bereich: ROM-BIOS, BASIC, Diagnoseroutinen
E 0 0 0	Cartridge-ROM-Bereich
D 0 0 0	Cartridge-ROM-Bereich
C 0 0 0	BIOS-Erweiterungen (XT-Festplattenstation)
B 0 0 0	Bildschirmspeicherbereich (bei PC, XT und AT)
A 0 0 0	erweiterter Bildschirmspeicherbereich
9 0 0 0	Hauptspeicher-RAM, bis 640 Kbyte
8 0 0 0	Hauptspeicher-RAM, bis 576 Kbyte
7 0 0 0	Hauptspeicher-RAM, bis 512 Kbyte
6 0 0 0	Hauptspeicher-RAM, bis 448 Kbyte
5 0 0 0	Hauptspeicher-RAM, bis 384 Kbyte
4 0 0 0	Hauptspeicher-RAM, bis 320 Kbyte
3 0 0 0	Hauptspeicher-RAM, bis 256 Kbyte
2 0 0 0	Hauptspeicher-RAM, bis 192 Kbyte
1 0 0 0	Hauptspeicher-RAM, bis 128 Kbyte
0 0 0 0	Hauptspeicher-RAM, bis 64 Kbyte; üblicherweise durch Systemsoftware belegt

Tabelle 1-2 Das Konzept der Speicherblöcke im IBM PC.

immer einen zusammenhängenden Speicherbereich erfordert, werden keine Blöcke ausgelassen. Adressen innerhalb dieses 640-Kbyte-Bereiches, die höher sind, als der momentan physikalisch vorhandene Speicher, dürfen nicht verwendet werden. Dem Versuch eines Programmes, auf einen solchen Speicherbereich zuzugreifen, steht kein fest definiertes Ergebnis gegenüber. Es ist durchaus möglich, daß keine Fehlermeldung erfolgt und das Programm weiterläuft, ja der Benutzer eventuelle Fehler gar nicht bemerkt.

Alle IBM PCs haben zumindest im ersten Block (0-Block) Speicherbausteine installiert. Der Mindestspeicher, der in allen Modellen vorhanden ist, beträgt 64 Kbyte, meist ist aber wesentlich mehr Kapazität anzutreffen. Die niedrigsten Adressen im 0-Block sind "traditionell" für die Systemsoftware reserviert. Dort werden solche Dinge wie Statusinformationen, Adreßtabellen, Zeichentabellen und Betriebssystemroutinen abgelegt. In Kapitel 3 wird der untere Speicherbereich näher untersucht.

Der A-Block ist für die Erweiterung des Bildschirmspeichers vorgesehen. Er wird von IBMs *Enhanced Graphics Adapter* (EGA) und dem *Professional Graphics Adapter* benötigt. Dieser Speicherbereich ist mit zahllosen Eigentümlichkeiten und Überraschungen versehen, so daß nur sehr wenige verläßliche Informationen über ihn bestehen. Am besten verstehen Sie den A-Block einfach als einen provisorischen Zwischenspeicher im Zusammenhang mit den erweiterten Bildschirmmodi.

Der B-Block wird gewöhnlich zum Aufnehmen des "normalen" Bildschirmspeichers benutzt. Er ist in zwei Hälften zu je 32 Kbyte aufgeteilt, deren Anfangsadressen hex B0000 und B8000 sind. Häufig wird zur Abkürzung einfach B0 bzw. B8 gesagt. Der Monochromadapter von IBM, eine Zusatzkarte, die den S/W-Monitor bedient, verfügt über 4 Kbyte des Speichers und liegt am Beginn des B0-Bereichs (die verbleibenden 28 Kbyte bleiben ungenutzt). Der 16 Kbyte große Speicherplatz des IBM Farb-/Grafikadapters, der die meisten anderen Monitore steuert und wie der Monochromadapter eine Zusatzkarte ist, liegt hingegen am Anfang des B8-Bereichs (auch hier werden die verbleibenden 16 Kbyte der Blockhälfte nicht genutzt).

Da die Aufteilung des B-Blocks bei IBM zu einem Standard geworden ist, ist es nützlich, zu wissen, daß der B0-Halbblock die Informationen für den Monochromadapter und der B8-Halbblock die für den Farb-/Grafikadapter enthalten. Für alle Modelle kann man sagen, daß der B-Block (zumindest scheinbar) zur Speicherung der Bildschirmdaten benutzt wird. Das gilt auch für den *Enhanced Graphics Adapter* und den *Professional Graphics Adapter*.

Der C-Block ist für zusätzlichen ROM-Speicher reserviert. Zunächst wurde dieser Bereich zur Speicherung der ROM-BIOS-Routinen, die der Steuerung der Festplattenstation dienen, die erstmals im XT vorhanden ist, verwendet. Eine Festplatte kann aber auch an den Standard PC angeschlossen werden. Diese Routinen wurden nicht an den Anfang oder das

Ende des Blockes gesetzt, sondern mitten hinein, die Startadresse liegt bei hex C8. Wir können wohl annehmen, daß auch alle zukünftigen BIOS-Erweiterungen in diesem Bereich gespeichert werden, vor allem jene, die neue Hardware unterstützen.
ROM-Speicher als Software-Cartridges, die mit dem PCjr auf dem US-Markt erschienen, erhalten als Speicherplatz die Blöcke D und E zugeordnet. Cartridge-Unterstützung wird kann fast jedem Modell hinzugefügt werden, wird jedoch praktisch nie ausgenutzt (bis eben auf den PCjr). Die Cartridge-Unterstützung wird in einem der folgenden Blöcke abgelegt: D0, D8, E0 und E8.
Fest installierte ROM-Programme werden für gewöhnlich in den F-Block geladen, dazu gehören unter anderem auch das ROM-Kassetten-BASIC, die ROM-BIOS- und die Test- und Diagnoseroutinen.

1.5 Konzeption des PC

Bevor wir zum nächsten Kapitel übergehen, wollen wir uns mit dem technischen Entwurf des IBM PC beschäftigen. Das verhilft uns zu einem besseren Verständnis darüber, was wichtig ist - und was nicht.
Ein Teil der IBM PC Konzeption dreht sich um die BIOS-Dienstleistungsroutinen (siehe auch Kapitel 8 bis 13). Sie stellen all jene Steuerungs- und Kontrollfunktionen bzw. -operationen zur Verfügung, die die CPU im Rahmen einer sinnvollen Arbeit unabdingbar benötigt. Das Motto, das hier zutrifft, lautet salopp, aber einprägsam formuliert: "Laß es das BIOS tun, mische dich nicht unnötig ein, dadurch kommt nur alles durcheinander." Dieses Konzept birgt viele Vorzüge. Es unterstützt gutes und übersichtlich strukturiertes Programmieren und verhindert einen Wust an speziellen Tricks "mit Selbstüberlistung", die schon der Fluch vieler Computer (und auch Programmierer!) waren. Dieses Konzept steigert auch die Software-Kompatibilität der unterschiedlichen PC-Modelle zueinander. Nicht zuletzt ist natürlich durch die damit gegebene Flexibilität der Anwendersoftware die Weiterentwicklung und Ausweitung der PC-Reihe gesichert. Das bedeutet nun nicht, daß es im Einzelfall nicht gute Gründe geben mag, die Hardwarekontrolle durch ein selbstgeschriebenes Programm zu übernehmen, aber das ist jedenfalls eine recht kniffelige Angelegenheit. Und schließlich, wenn ein BIOS vorhanden ist, das uns alle grundlegenden Funktionsroutinen zur Verfügung stellt, warum sollten wir es dann nicht nutzen?
Haben Sie aber dennoch vor, Ihrem Programm die Hardwarekontrolle zu übergeben, sollten Sie verstanden haben, daß der Mechanismus dazu in der Benutzung der Ports liegt (nähere Erläuterungen zu den Ports finden Sie in Kapitel 2). Die einzige Ausnahme besteht in der Ausgabe auf den Bildschirm: In diesem Fall arbeitet der PC mit einem Speicherkonzept, nicht mit Ports. Ansonsten aber wird die gesamte direkte Steuerung und

Kontrolle der Hardware mit Hilfe der Ports abgewickelt. Von einigen wenigen Ausnahmen abgesehen wird das Entwurfskonzept der PC-Familie durch die direkte Benutzung der Ports durch externe Software verletzt, so daß davon abzuraten ist. Die erwähnten Ausnahmen dieser Regel finden wir in den Elementen, für die IBM erst gar keine BIOS-Kontrolle vorgesehen hat; hier ist vor allem die Tonerzeugung zu erwähnen (siehe auch Kapitel 7).

Kapitel 2
Interne Kommunikation

2.1 Kommunikation des 8088 23
2.1.1 8088 Datenformat 24
2.2 Speicheradressierung des 8088 25
2.2.1 Speichererweiterung durch segmentierte Adressen 25
2.2.2 Die 14 Register des 8088 26
2.2.2.1 Zwischenspeicherregister 27
2.2.2.2 Segmentregister 29
2.2.2.3 Offsetregister 31
2.2.2.4 Indexregister 31
2.2.2.5 Flaggenregister 32
2.2.2.6 Speicheradressierung mit Hilfe von Registern 32
2.2.2.7 Regeln zur Benutzung der Register 34
2.3 Portbenutzung des 8088 35
2.4 Unterschiede in der Portbenutzung 36
2.5 Verarbeitung von Interrupts 36
2.6 Stapel 38
2.7 Umgekehrtes Speichern von Worten 40

Je besser die eigenen Programmierkenntnisse sind, desto eher erkennt man die Grenzen der Programmiersprachen. Hochentwickelte Sprachen, wie BASIC oder C, stellen einfach nicht alle möglichen Funktionen zum Programmieren bereit, so daß es vorkommen kann, daß bei Spezialanwendungen Lücken im Sprachumfang entdeckt werden. Manchmal werden Sie tiefer in Ihr System "eindringen" und einige der Routinen, die von den Programmiersprachen verwendet werden, direkt benutzen wollen. Oder Sie gehen sogar noch weiter und programmieren auf der Hardwareebene.
Manche Sprachen stellen begrenzte Mittel zur Verfügung, um direkt mit dem Speicher (PEEK und POKE in BASIC) oder sogar mit anderen Bausteinen (INP und OUT in BASIC) zu kommunizieren. Trotz der gegebenen Möglichkeit greifen die meisten Programmierer aber auf die Assemblersprache zurück, die Grundsprache, aus der alle anderen Programmiersprachen und auch das Betriebssystem aufgebaut sind. Die 8088-Assemblersprache besteht, wie alle anderen Assemblersprachen, aus einem Befehlssatz mit symbolischen und mnemonischen Codes. Die Codes und die damit verknüpften Daten werden in eine binäre Form, die Maschinensprache, umgewandelt und im Speicher abgelegt. In diesem Zustand können die Daten durch die Computerbausteine übertragen werden und die ihnen übergebene Aufgabe ausführen.
Die Befehlsliste des 8088 kann in verschiedene Kategorien unterteilt werden. Zum ersten gibt es Befehle für einfache arithmetische Operationen mit 8- und 16-bit-Ganzzahlen. Andere Befehle dienen dem Datentransfer und wieder andere können einzelne Bits manipulieren. Außerdem existieren Befehle, die logische Entscheidungen aufgrund vorher überprüfter Werte treffen können. Und schließlich erlauben einige Assemblerbefehle die Steuerung und Kontrolle der Hardware. Die Länge der Befehle variiert zwischen einem und sechs Bytes, wobei die am häufigsten genutzten Befehle die kürzesten Codes aufweisen.
Die Programmierung in Assembler ist auf zwei Ebenen möglich: Entweder man entwickelt eine Schnittstellenroutine, die die Programme in einer höheren Programmiersprache an die niederen DOS- und ROM-BIOS-Routinen anbindet; oder aber man schreibt vollkommen unabhängige Assemblerprogramme, die dann natürlich wirklich exotische Aufgaben auf der Hardwareebene übernehmen müssen. Um Assemblerprogrammierung zu verstehen, müssen Sie über die Verarbeitung von Daten im 8088 und den Austausch von Daten zwischen dem Prozessor und dem Rest des Computers Bescheid wissen. Im Mittelpunkt dieses Kapitels steht daher die computerinterne Kommunikation.

Mnemonik	Bedeutung	Erklärung
AAA	ASCII adjust for addition	ASCII-Korrektur für Addition
AAD	ASCII adjust for division	ASCII-Korrektur für Division
AAM	ASCII adjust for multiplication	ASCII-Korrektur für Multiplikation
AAS	ASCII adjust for subtraction	ASCII-Korrektur für Subtraktion
ADC	Add with carry	Addition mit Übertrag
ADD	Add	Addition
AND	AND	AND-Funktion
CALL	CALL	Aufruf
CBW	Convert byte to word	Byte-Wort-Konvertierung
CLC	Clear carry flag	Übertragsflagge rücksetzen
CLD	Clear direction flag	Richtungsflagge rücksetzen
CLI	Clear interrupt flag	Interrupt-Flagge rücksetzen
CMC	Complement carry flag	Übertragsflagge komplementieren
CMP	Compare	Vergleich
CMPS	Compare byte or word (of string)	Stringvergleich (byte- oder wortweise)
CMPSB	Compare byte string	Stringbyte vergleichen
CMPSW	Compare word string	Stringwort vergleichen
CWD	Convert word to double word	Word-Doppelwort-Konvertierung
DAA	Decimal adjust for addition	Dezimalkorrektur bei Addition
DAS	Decimal adjust for subtraction	Dezimalkorrektur bei Subtraktion
DEC	Decrement	Dekrementieren
DIV	Divide	Division
ESC	Escape	Aussprung
HLT	Halt	Programmhalt
IDIV	Integer divide	Ganzzahldivision
IMUL	Integer multiply	Ganzzahlmultiplikation
IN	Input byte or word	Byte- oder Wort-Eingabe
INC	Increment	Inkrementieren
INT	Interrupt	Interrupt-Aufruf
INTO	Interrupt on overflow	Interrupt bei Überlauf
IRET	Interrupt return	Interrupt-Rücksprung
JA	Jump on above	Sprung bei oberhalb
JAE	Jump on above or equal	Sprung bei oberhalb/gleich
JB	Jump on below	Sprung bei unterhalb
JBE	Jump on below or equal	Sprung bei unterhalb/gleich
JC	Jump on carry	Sprung bei gesetzter Übertragsflagge
JCXC	Jump on CX zero	Sprung wenn CX gleich 0
JE	Jump on equal	Sprung bei gleich
JG	Jump on greater	Sprung bei größer
JGE	Jump on greater or equal	Sprung bei größer/gleich
JL	Jump on less than	Sprung bei kleiner
JLE	Jump on less than or equal	Sprung bei kleiner/gleich
JMP	Jump	Sprung
JNA	Jump on not above	Sprung bei nicht oberhalb
JNAE	Jump on not above or equal	Sprung bei nicht oberhalb oder gleich
JNB	Jump on not below	Sprung bei nicht unterhalb
JNBE	Jump on not below or equal	Sprung bei nicht unterhalb oder gleich
JNC	Jump on not carry	Sprung bei rückgesetzter Übertragsflagge
JNE	Jump on not equal	Sprung bei ungleich
JNG	Jump on not greater	Sprung bei nicht größer
JNGE	Jump on not greater or equal	Sprung bei nicht größer oder gleich

Tabelle 2-1 Der Befehlssatz des 8088 Mikroprozessors

Mnemonik	Bedeutung	Erklärung
JNL	Jump on not less than	Sprung bei nicht kleiner
JNLE	Jump on not less than or equal	Sprung bei nicht kleiner oder gleich
JNO	Jump on not overflow	Sprung bei rückgesetzter Überlaufsflagge
JNP	Jump on not parity	Sprung bei rückgesetzter Paritätsflagge
JNS	Jump on not sign	Sprung bei rückgesetzter Vorzeichenflagge
JNZ	Jump on not zero	Sprung bei rückgesetzter Nullflagge
JO	Jump on overflow	Sprung bei gesetzter Überlaufsflagge
JP	Jump on parity	Sprung bei gesetzter Paritätsflagge
JPE	Jump on parity even	Sprung bei gerader Parität
JPO	Jump on parity odd	Sprung bei ungerader Parität
JS	Jump on sign	Sprung bei gesetzter Vorzeichenflagge
JZ	Jump on zero	Sprung bei gesetzter Nullflagge
LAHF	Load AH with flags	AH mit Flaggenstatus laden
LDS	Load pointer into DS	Zeiger in DS Laden
LEA	Load effective address	Effektive Adresse laden
LES	Load pointer into ES	Zeiger in ES laden
LOCK	LOCK bus	Bus blockieren
LODS	Load byte or word (of string)	String-Byte oder -Wort laden
LODSB	Load byte (string)	String-Byte laden
LODSW	Load word (string)	String-Wort laden
LOOP	LOOP	Schleife
LOOPE	LOOP while equal	Schleifendurchlauf bei gleich
LOOPNE	LOOP while not equal	Schleifendurchlauf bei ungleich
LOOPNZ	LOOP while not zero	Schleifendurchlauf bei ungleich Null
LOOPZ	LOOP while zero	Schleifendurchlauf bei Null
MOV	Move	Verschieben
MOVS	Move byte or word (of string)	String-Byte oder -Wort-Verschiebung
MOVSB	Move byte (string)	String-Byte-Verschiebung
MOVSW	Move word (string)	String-Wort-Verschiebung
MUL	Multiply	Multiplikation
NEG	Negate	Negation
NOP	No operation	Keine Operation
NOT	NOT	NOT-Funktion
OR	OR	OR-Funktion
OUT	Output byte or word	Ausgabe (Byte oder Wort)
POP	POP	Rücknahme vom Stapel
POPF	POP flags	Rücknahme der Statusflaggen vom Stapel
PUSH	PUSH	Übergabe auf den Stapel
PUSHF	PUSH flags	Übergabe der Statusflaggen auf den Stapel
RCL	Rotate through carry left	Linksrotation (einschließlich Überlauf)
RCR	Rotate through carry right	Rechtsrotation (einschließlich Überlauf)
REP	Repeat	Wiederholung
RET	Return	Rücksprung
ROL	Rotate left	Linksrotation
ROR	Rotate right	Rechtsrotation
SAHF	Store AH into flags	AH in Flaggen speichern
SAL	Shift arithmetic left	Arithmetische Linksverschiebung
SAR	Shift arithmetic right	Arithmetische Rechtsverschiebung
SBB	Subtract with borrow	Subtraktion mit umgekehrtem Übertrag
SCAS	Scan byte or word (of string)	Byte oder Wort suchen (String)
SCASB	Scan byte (string)	Byte suchen (String)
SCASW	Scan word (string)	Wort suchen (String)

Tabelle 2-1 (Forts.) Der Befehlssatz des 8088 Mikroprozessors

Mnemonik	Bedeutung	Erklärung
SHL	Shift left	Linksverschiebung
SHR	Shift right	Rechtsverschiebung
STC	Set carry flag	Übertragsflagge setzen
STD	Set direction flag	Richtungsflagge setzen
STI	Set interrupt flag	Interruptflagge setzen
STOS	Store byte or word (of string)	Byte oder Wort (eines Strings) speichern
STOSB	Store byte (string)	Byte eines Strings speichern
STOSW	Store word (string)	Wort eines Strings speichern
SUB	Subtract	Subtraktion
TEST	TEST	Test
WAIT	WAIT	Warten
XCHG	Exchange	Austausch
XLAT	Translate	Übersetzung
XOR	Exclusive OR	Exklusiv-Oder-Funktion

Tabelle 2-1 (Forts.) Der Befehlssatz des 8088 Mikroprozessors

2.1 Kommunikation des 8088

Der 8088 verfügt über drei Möglichkeiten, Bausteine zu beeinflussen: Per direktem und indirektem Speicherzugriff, durch Ports und mit den Interruptbefehlen.

Speicher wird benutzt, indem Daten in einzelne Speicherzellen eingeschrieben oder aus ihnen ausgelesen werden. Mit Hilfe numerischer Adressen sind alle Speicherstellen jederzeit auffindbar. Der Speicherzugriff ist direkt, das heißt, unter Zuhilfenahme des 8237A-Bausteines (auch als DMA-Controller bekannt) oder indirekt, mit den internen Registern des 8088, möglich. Die Diskettenlaufwerke und die seriell übertragenden Kommunikationsports sind in der Lage, direkt auf den Speicher zuzugreifen, sie benutzen dazu den DMA-Controller. Alle anderen Geräte transferieren Daten vom und zum Speicher mit Hilfe der Register des 8088. Weitere Informationen über den DMA-Controller finden Sie in Kapitel 1.2.2, mehr über Register in Kapitel 2.2.2.

Ports sind die Grundbestandteile des 8088 zur Kommunikation mit anderen Bauteilen außer dem Speicher. Ähnlich wie Speicherstellen werden auch Ports durch Nummern gekennzeichnet und Daten können ein- und ausgelesen werden. Die Portspezifikationen innerhalb der PC-Modellreihe sind bis auf wenige Ausnahmen einheitlich (Kapitel 2.5 gibt Ihnen hierzu nähere Auskünfte).

Interrupts (manchmal auch *Unterbrechungen* genannt) sind Signale, die dem Prozessor eine Mitteilung machen. So löst z.B. der Druck auf eine Taste der Tastatur einen Interrupt aus, der zur CPU gesendet wird und dort die erwartete Reaktion veranlaßt. Die Interrupts sind für den 8088 von großer Bedeutung, da sie die Basis für Wechselwirkungen mit seiner Umgebung darstellen. Das Konzept der Interrupts ist aber auch für ande-

re Zwecke hervorragend geeignet. So können beispielsweise das BIOS oder das Betriebssystem Softwareinterrupts erzeugen, um spezielle Dienstleistungsroutinen aufzurufen und auszuführen. Die Interrupts sind zur Programmierung des PC so wichtig, daß ihnen am Ende dieses Kapitels ein eigener Abschnitt gewidmet ist.

2.1.1 8088 Datenformat

Numerische Daten. Der 8088 kann nur mit vier einfachen Datenformaten arbeiten, deren Werte alle ganzzahlig sein müssen. Die Formate bauen sich aus zwei Grundeinheiten auf: dem Byte, bestehend 8 Bits und dem Wort, das 16 Bits oder 2 Bytes umfaßt. Beide sind auf die 16-bit-Verarbeitung des 8088 und seines 8-bit-Datenbusses zurückzuführen. Ein Byte ist die Grundeinheit des Prozessors, soweit es um die Adressierung geht; wenn der 8088 Speicherplätze adressiert, sind es einzelne Bytes, die angesprochen werden. Ein Byte kann vorzeichenlose Zahlenwerte von 0 bis 255 (2^8 Möglichkeiten) darstellen. Haben wir es mit vorzeichenbehafteten Zahlen zu tun, das heißt, mit positiven und negativen Zahlen, kann ein Byte Werte von -128 bis +127 speichern.
Für größere Zahlenwerte behandelt der 8088 einfach zwei Bytes als eine Einheit. Das 2-byte-Wort ist die geläufigste Form, obwohl auch größere Worte theoretisch möglich sind. Ein 2-byte-Wort, das vorzeichenlos interpretiert wird, verfügt über einen Wertebereich zwischen 0 und 65.535. Mit Vorzeichen, das heißt, bei sowohl positiven als auch negativen Zahlen, beläuft sich der Wertebereich von -32.768 bis +32.767.

Länge	Vorzeichen	Bereich	
		Dez	Hex
8	Nein	0 – 255	00 – FF
8	Ja	– 128 – 00 – + 127	80 – 00 – 7F
16	Nein	0 – 65.535	0000 – FFFF
16	Ja	– 32.768 – 0 – + 32.767	8000 – 0000 – 7FFF

Tabelle 2-2 Die vier Datenformate des 8088

Zeichendaten. Zeichendaten werden im ASCII-Format gespeichert. Jedes Zeichen hat eine Länge von genau einem Byte. Da der 8088 ASCII-Zeichen nicht als solche erkennt, behandelt er diese Bytes wie alle anderen auch - mit einer Ausnahme: Der Befehlssatz erlaubt die Addition und Subtraktion von ASCII-Dezimalzahlen. Die eigentliche Arithmetik wird dabei binär durchgeführt, aber die AF-Flagge (siehe Kapitel 2.2.2.6) in Verbindung mit einigen wenigen Spezialbefehlen ermöglicht es, mit dezimalen Zeichen zu arbeiten und auch dezimale Ergebnisse zu erhalten.
In Anhang C finden Sie mehr Informationen über ASCII-Zeichen und den erweiterten ASCII-Zeichensatz des PC.

2.2 Speicheradressierung des 8088

Der 8088 ist als 16-bit-Prozessor nicht in der Lage, direkt mit Zahlen, deren Länge 16 Bit übersteigt, zu arbeiten. Der größte darstellbare Wert beträgt somit 65.535. Theoretisch kann der 8088 also nur auf einen Speicherbereich von maximal 64 Kbyte zugreifen. Wie Sie im vorigen Kapitel erfahren haben, kann er aber tatsächlich bis zu 1.024 Kbyte adressieren. Diese Fähigkeit basiert auf dem 20-bit-Datenbus, der die Speicheradressierung von 2^{16} (65.535) auf 2^{20} (1.048.576) Möglichkeiten erweitert. Trotzdem ist der 8088 durch seine 16-bit-Struktur eingeschränkt, da das 20-bit-Format in ein für den Prozessor verarbeitbares 16-bit-Format umgewandelt werden muß.

2.2.1 Speichererweiterung durch segmentierte Adressen

Der 8088 teilt den adressierbaren Speicher in eine willkürliche Anzahl von Segmenten, von denen keines mehr als 64 Kbyte enthält. Jedes Segment beginnt an einer Stelle, deren Wert, durch 16 geteilt, eine ganze Zahl ergibt. Diesen Speicherplatz bezeichnet man als *Segmentadresse* oder *Segmentparagraph*. Um einzelne Bytes oder Worte anzusprechen, brauchen wir eine Unteradresse, die *Offsetadresse* genannt wird und auf eine bestimmte Stelle innerhalb eines Segmentes zeigt. Die Segmente werden durch die Segmentadressen gekennzeichnet. Offsetadressen werden immer relativ zum Beginn eines Segmentes, das heißt, relativ zur Segmentadresse, berechnet und heißen daher auch *Relativadressen* oder *relativer Offset*.
Durch die Kombination einer 16-bit-Segmentadresse und einer 16-bit-Relativadresse werden die 20 Bit langen Speicheradressen erzeugt. Die Segmentadresse wird dabei um vier Bits nach links verschoben und zum relativen Offset addiert, so daß sich eine vollständige 20-bit-Adresse ergibt; zusammengenommen werden die zwei 16-bit-Adressen oft *segmentierte Adressen* oder auch *Vektor* genannt, letzteres vor allem, wenn sie zu Interrupts gehören (mehr über Interruptvektoren in Kapitel 2.5).
Segmentadressen werden als fünfstellige Hexadezimalzahlen geschrieben und haben immer eine 0 an der letzten Stelle (Beispiel: FFE40 oder B8120). Die Null am Ende der Hexzahl beruht auf der Multiplikation der ursprünglichen 16-bit-Zahl, die einer vierstelligen Hexzahl entspricht, mit 16. Denselben Verschiebungseffekt erhalten wir bei der Multiplikation eines Dezimalwertes mit seiner Basis, der Zahl 10, beispielsweise 10 x 23 ergibt 230. Die Tatsache, daß der Segmentadressenteil um 4 Bits nach links verschoben ist (Multiplikation mit 16), ist auch der Grund, weshalb er nur auf Speicheradressen zeigen kann, die ein ganzzahliges Vielfaches von 16 sind. Zur genauen Definition der Speicherplätze in einem Segment werden die Relativadressen benötigt, die man als vierstellige Hexzahlen darstellen kann.

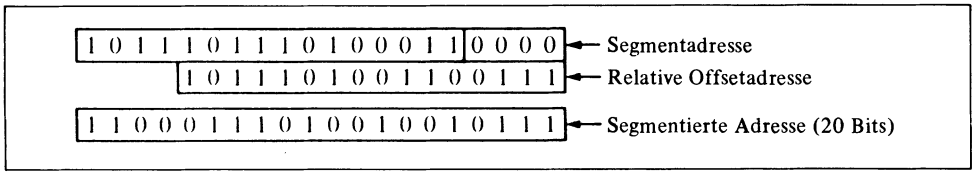


Bild 2-1 Aus Segmentadresse (die auf das 64-KB-Segment zeigt) und relativer Offsetadresse wird eine segmentierte 20-bit-Adresse gebildet.

Die Segmentadresse, die auf den Anfang eines Speichersegmentes (64 Kbytes) zeigt und die Relativadresse, die auf ein bestimmtes Byte innerhalb des Segments zeigt, werden durch die CPU zu einer 20-bit-Adresse kombiniert
In der Kombination ergeben die zwei Zahlen eine fünfstellige Hexzahl, die in eine 20-bit-Adresse umgewandelt wird. Nehmen wir z.B. die hexadezimale Segmentadresse 1234 und multiplizieren sie mit 16, wir erhalten 12340. Addieren wir hierzu den relativen Offset des gesuchten Bytes, beispielsweise 4321, bekommen wir die fünfstellige Hexzahl:

12340 Segmentadresse in Hexnotation, um 4 Bits nach links verschoben.
+4321 Offsetadresse in Hexnotation.
16661 20-bit-Segmentadresse in Hexnotation.

Die Schreibweise für eine segmentierte Adresse (Segmentadresse und Relativadresse) sieht folgendermaßen aus: 0000:0000, wobei die Segmentadresse links vom Doppelpunkt, die Relativadresse rechts davon steht. Die 20-bit-Adresse FFE6E kann in segmentierter Schreibweise als FFE4:002E ausgedrückt werden. Dabei läßt sich eine einzige 20-bit-Adresse auf verschiedene Arten in segmentierter Notation ausdrücken, abhängig von der jeweils gewählten Segmentadresse.

2.2.2 Die 14 Register des 8088

Der 8088 wurde entworfen, um Befehle und arithmetische und logische Operationen zur gleichen Zeit auszuführen, in der er Instruktionen erhält und der Datenaustausch mit dem Speicher vonstatten geht. Dafür benötigt er 16-bit-Register.
Insgesamt verfügt der Prozessor über 14 Register. Jedem Register ist eine spezielle Aufgabe zugeteilt: vier Zwischenspeicherregister, die zur temporären Speicherung von Zwischenresultaten und Operanden arithmetischer und logischer Funktionen benutzt werden; vier Segmentregister, die die Startadressen der verschiedenen Speichersegmente aufnehmen können; fünf Zeiger- und Indexregister, die die Offsetadressen zwischenspeichern

und mit den Segmentadressen zusammen die Speicherplätze definieren. Das letzte Register ist ein Flaggenregister, das aus neun 1-bit-Flaggen besteht, die die Statusinformationen und Kontrolloperationen des 8088 festhalten.

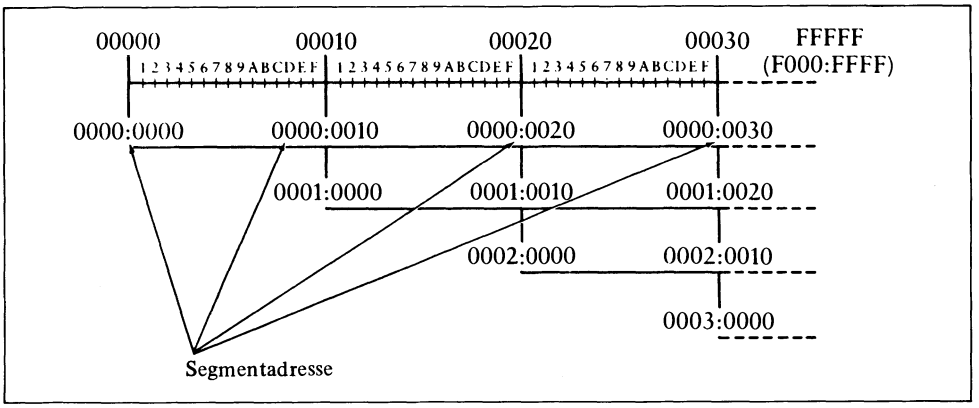


Bild 2-2 Die Offsetadresse bezieht sich stets auf eine Segmentadresse. Daher kann ein bestimmter Speicherplatz mit unterschiedlichen segmentierten Adressen bezeichnet werden.

2.2.2.1 Zwischenspeicherregister

Den größten Teil der Abarbeitungszeit eines Programmes verbringt die CPU mit dem Datentransfer vom und zum Speicher. Die Zugriffszeit kann wesentlich verringert werden, indem oft benutzte Operanden und Resultate im 8088 selbst abgespeichert werden. Die vier 16-bit-Register, die allgemein *Zwischenspeicherregister* oder *Datenregister* genannt werden, sind zu diesem Zweck eingebaut worden.
Die Datenregister werden mit AX, BX, CX und DX bezeichnet. Jedes Register kann unterteilt und als 8-bit-Halbregister adressiert und eingesetzt werden. Die höherwertigen Halbregister werden mit AH, BH, CH und DH bezeichnet, die niederwertigen mit AL, BL, CL und DL. Der Gebrauch als Halb- oder Vollregister kann je nach Notwendigkeit gemischt geschehen.
Das Hauptaufgabengebiet der Zwischenspeicherregister liegt bei den arithmetischen Operationen. Dort werden sie als Zwischenspeicher genutzt. Addition und Subtraktion sind auch in den Speichern möglich, die Verarbeitungszeit ist in diesem Fall aber größer.
Trotz der universellen Einsatzmöglichkeit der Datenregister sind gewisse Aufgaben auf bestimmte Register beschränkt.
Das AX-Register ist ein Akkumulator und wird hauptsächlich bei arithmetischen Operationen verwendet.

Das BX-Register, auch *Basisregister* genannt, wird oft als Zeiger auf Tabellen im Speicher eingesetzt. Es findet aber auch als Speicherplatz für den Offsetteil einer segmentierten Adresse Verwendung.
Das CX-Register wird häufig als Zählregister zur Schleifensteuerung benutzt. Der LOOP-Befehl in BASIC verwendet z.B. dieses Register, um die Anzahl der Schleifendurchläufe zu zählen. Keines der anderen Register kann diese Aufgabe übernehmen.
Das DX-Register kann 16-bit-Daten für beliebige Zwecke speichern.

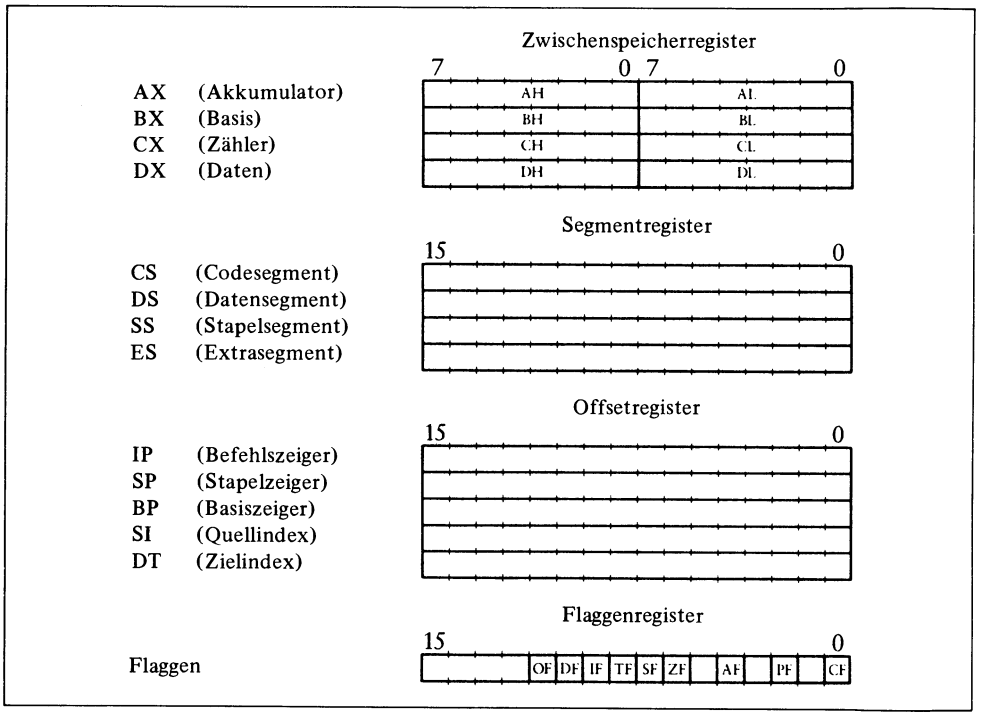


Bild 2-3 Die Register und Flaggen des 8088

Die Datenregister des 8088 sind trotz ihrer Funktion als Daten- und Operandenzwischenspeicher auch für andere Zwecke einsetzbar. Relative Offsetadressen beispielsweise werden oft als Programmparameter gefordert, sie können in diesem Fall auch in den Registern abgelegt werden.

2.2.2.2 Segmentregister

Die komplette Adresse einer Speicherstelle besteht aus der Adresse des Speichersegments und der relativen Offsetadresse. Es gibt vier Register, die vier spezielle 64-Kbyte-Segmente anwählen. Von diesen vier Registern sind drei auf die Ausführung besonderer Aufgaben zugeschnitten:
Das CS-Register lokalisiert das Segment, in welchem das gerade ablaufende Programm gespeichert ist, das sogenannte *Codesegment*.
Das DS-Register zeigt auf das Datensegment. Das ist das Segment, in dem die aktuellen Daten gespeichert sind.
Das SS-Register verweist auf das Stapelsegment, einem temporären Arbeitsbereich, der die Daten des laufenden Programmes verwaltet. In Kapitel 2.6 finden Sie mehr über den Stapel.
Das vierte Segmentregister, ES, zeigt normalerweise auf ein Segment, das üblicherweise zur Unterstützung des Datensegmentes herangezogen wird, so daß der Datenspeicherbereich nicht auf 64 Kbyte beschränkt ist. Außerdem wird es für den Datenaustausch zwischen den einzelnen Segmenten eingesetzt.
Es ist nicht ungewöhnlich, daß sich die vier Segmente teilweise oder sogar ganz überlappen. In der Regel wird auch nur ein Teil des Arbeitsbereiches in den Segmenten genutzt. So kann es vorkommen, daß nur 16 Kbyte eines 64-Kbyte-Segmentes für eine bestimmte Aufgabe verwendet wird, der Rest des Segmentes bleibt ungenutzt.

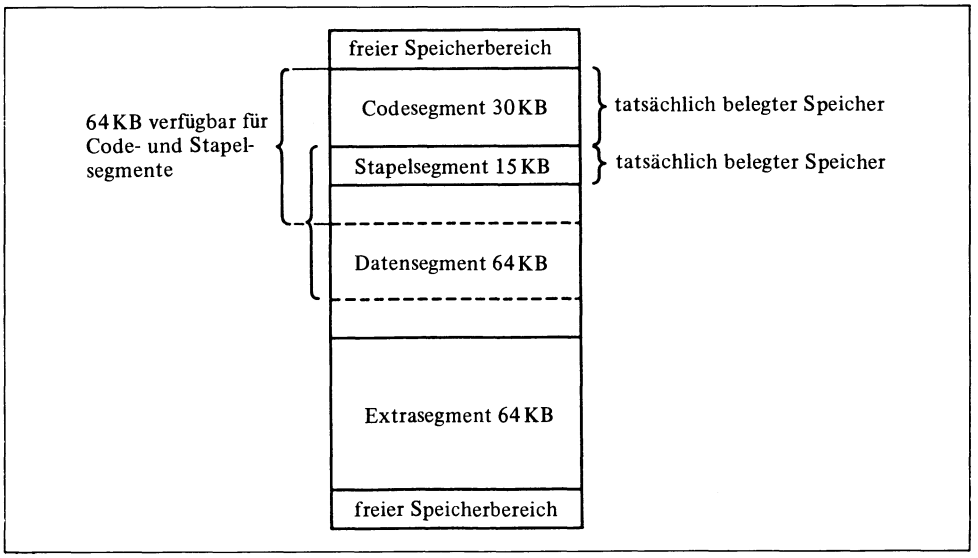


Bild 2-4 Speichersegmente können getrennt sein oder überlappen. Der Beginn von Stapel-, Daten- und Extrasegment wird durch die entsprechenden Adressen in den Segmentregistern (SS, DS oder ES) festgelegt.

Alle 8088-Befehle, die den Speicher benutzen, verwenden automatisch das geeignete Segmentregister. Der MOV-Befehl, sofern er sich auf Daten bezieht, spricht das DS-Register an. Der JMP-Befehl, der Auswirkungen auf den Programmfluß hat, zieht das CS-Register heran. Meistens kann das automatisch angesprochene Register durch ein anderes ersetzt werden. In der Assemblersprache geschieht das mit den *Segment-override-Präfixen.*

Mit dem Verständnis der Segmentregister bekommen Sie gleichzeitig Einblick in die Grenzen einer 16-bit-Architektur bezüglich der Speicheradressierung. Erfordert ein Programm oder ein Datenblock mehr als 64 Kbyte Speicherkapazität, sind einige spezielle Manipulationen der Segmentregister unerläßlich. Dazu nachfolgend einige Hinweise.

Wird das CS-Register nicht verändert, beträgt die maximale Gesamtlänge unserer Programme 64 Kbyte. Da der 8088 entwickelt wurde, um in jedem Fall die Kontrolle über ein Programm zu behalten, ist es sehr schwierig, das CS-Register direkt zu beeinflussen und damit die Codesegmentadresse zu verändern. Bestimmte 8088-Befehle, wie weite Sprünge und weite Aufrufe, erlauben es, das CS-Register indirekt auf den neuesten Stand zu bringen. Auf diese Weise ist es in vielen Programmiersprachen möglich, einen wesentlich größeren Speicherbereich für das Programm bereitzustellen. Das gilt nur für die Version 2 des Microsoft C-Compilers und für Pascal, nicht für Version 1 und interpretierendes BASIC.

Andererseits ist es sehr einfach, das DS-Register oder das ES-Register zu manipulieren, und damit einen größeren Speicherbereich für Daten bereitzustellen. Obgleich damit die theoretische Möglichkeit besteht, einen nahezu unbegrenzten Datenspeicher festzulegen, können die meisten Programmiersprachen aufgrund ihres Aufbaus nur einen Datenspeicherbereich von maximal 64 Kbyte nutzen. Das wird aber nur selten zum Problem, da für die Mehrzahl aller Programme 64 Kbyte völlig ausreichend ist. Außerdem ist es eine Binsenweisheit der programmierenden Zunft, daß zu viel Speicherplatz genauso schädlich für die Programmerstellung ist wie zu wenig.

Hinweis: Im interpretierenden BASIC sieht die Verteilung der Segmentregister etwas anders aus. Das CS-Register zeigt auf den BASIC-Interpreter. Ein BASIC-Programm und dessen Daten sind aus der Sicht des 8088 vollkommen gleichwertig, beide haben einen Datenstatus und benutzen daher das DS-Register. Aus diesem Grund ergibt sich für ein BASIC-Programm und dessen Daten zusammengenommen eine Gesamtspeicherkapazität von 64 Kbyte.

Für bestimmte BASIC-Operationen, wie etwa PEEK und POKE, stellt die DEF-SEG-Anweisung ein Äquivalent zum Setzen des DS-Registers dar. PEEK und POKE selbst werden zum Festlegen der Offsetadresse innerhalb des Datensegmentes verwendet. In Kapitel 3.2.2 wird erklärt, wie auf den echten DS-Wert auch von BASIC aus zugegriffen werden kann.

2.2.2.3 Offsetregister

Fünf Offsetregister werden zur genauen Bestimmung eines Bytes oder eines Wortes in einem 64-Kbyte-Segment benötigt. Ein Register, der Befehlszeiger, deutet auf den momentan auszuführenden Befehl im Codesegment. Zwei Stapelregister sind eng mit dem Stapel, einer Art Ablagespeicher verbunden (mehr über den Stapel finden Sie in Kapitel 2.6). Die verbleibenden zwei Register, das sind Indexregister, zeigen auf den benötigten Operanden im Datensegment.
Der Befehlszeiger (IP für *Instruction Pointer* oder auch PC für *Programm Counter*) enthält die Offsetadresse des aktuellen Befehls im Codesegment. Gemeinsam mit dem CS-Register bestimmt er die Speicherstelle des nächsten auszuführenden Befehls.
Programme haben keine direkte Zugriffsmöglichkeit zum IP-Register. Es gibt aber eine Reihe von Befehlen, die den Befehlszeiger indirekt beeinflussen, beispielsweise JMP oder CALL, oder den aktuellen Stand sichern und auf den Stapel speichern.
Die Stapelregister unterteilen sich in den *Stapelzeiger* (*Stack Pointer*) oder kurz SP und den *Basiszeiger* (*Base Pointer*) oder kurz BP. Beide enthalten Offsets im Stapelsegment. Der SP zeigt auf die momentan höchste Stelle des Stapels. Der BP gibt eine "Momentaufnahme" der derzeitigen Stapelspitze wieder, so daß später jederzeit festgestellt werden kann, an welcher Stelle Daten und Informationen abgelegt worden sind. Der BP ist insbesondere zur Erstellung von Schnittstellenroutinen in Assembler sehr wichtig. Wir werden das anhand der Beispiele der Kapitel 8 bis 20 feststellen.

2.2.2.4 Indexregister

Die Indexregister, die auch Quellindex (SI für *Source Index*) und Zielindex (DI für *Destination Index*) genannt werden, arbeiten üblicherweise mit anderen Registern (AX, BX, CX und DX) oder einem Befehlsoffset, der den relativen Offset eines Datenfeldes im Datensegment darstellt, zusammen. Die beiden Indexregister enthalten dann die Offsets innerhalb des Datenfeldes. Eine häufige Anwendung besteht in der Übertragung von Datenketten (Strings) von einem Speicherbereich zu einem anderen. Die Stringbefehle, die diese Register ansprechen, transferieren die Daten als Bytes oder als Worte. Die Indexregister erhöhen ihren Wert normalerweise von selbst, das heißt, es muß nicht explizit addiert werden, um zur nächsten Speicherstelle zu gelangen.

2.2.2.5 Flaggenregister

Das vierzehnte und letzte Register des 8088 ist das Flaggenregister, eine Ansammlung einzelner Kontrollbits, die *Flaggen* (Flags) genannt werden. Die Flaggen sind zusammengefaßt in Form eines Registers zugänglich, sie können gespeichert oder zurückgesetzt, aber auch als gewöhnliche Daten betrachtet werden. Eigentlich werden die Flaggen aber als unabhängige 1-bit-Einheiten gesetzt und abgefragt.
Von den 16 zur Verfügung stehenden Bits sind sieben bedeutungslos, die restlichen neun können in zwei Gruppen unterteilt werden: sechs Statusflaggen, die die Prozessorstatusinformationen speichern, und drei Kontrollflaggen für einige der 8088-Befehle. Leider existiert bezüglich der Bezeichnungen der Flaggen keine Einheitlichkeit, das betrifft insbesondere auch die Notation für *gesetzt* und *nicht gesetzt*.

Code	Name	Verwendung (Flagge wird gesetzt, wenn ...)
CF	Übertragsflagge (Carry)	bei einer arithmetischen Operation ein Übertrag auftritt;
OF	Überlaufflagge (Overflow)	bei einer arithmetischen Operation ein Überlauf auftritt;
ZF	Nullflagge (Zero)	ein Ergebnis gleich Null oder ein Vergleich „gleich" ist;
SF	Vorzeichenflagge (Sign)	ein Ergebnis negativ oder ein Vergleich „ungleich" ist;
PF	Paritätsflagge (Parity)	die Anzahl der 1-Bits gerade ist;
AF	Hilfsübertragsflagge (Auxiliary Carry)	die Korrektur eines Ergebnisses in BCD-Arithmetik nötig ist.

Tabelle 2-3 Die sechs Statusflaggen im Flaggenregister des 8088

Code	Name	Verwendung
DF	Richtungsflagge (Direction)	Richtungskontrolle links/rechts in Wiederholfunktionen (z.B. Verwendung von SI und DI)
IF	Interruptflagge (Interrupt)	Interruptzulassung
TF	Einzelschrittflagge (Trap)	Einzelschrittbetrieb (z.B. in DEBUG) durch Softwarestopps nach jedem Befehl

Tabelle 2-4 Die drei Kontrollflaggen im Flaggenregister des 8088

2.2.2.6 Speicheradressierung mit Hilfe von Registern

Eine Speicherzelle wird immer durch eine Verknüpfung der Segmentadresse und des relativen Offsets adressiert; der Segmentteil einer Adresse kommt aus einem der vier Segmentregister.

Der Offsetteil kann aus einer, zwei oder drei der folgenden Quellen bezogen werden:

* Ein relativer Offsetwert des Befehls selbst;
* Ein Register, gewöhnlich AX, BX, CX oder DX;
* Ein Indexregister, SI oder DI.

Beim Erstellen oder Lesen eines Assemblerprogrammes sollten Sie stets bedenken, daß nicht jeder Befehl alle möglichen Offsetadressierungen erlaubt. Durch Ausprobieren können Sie herausfinden, welche Verknüpfungen möglich sind. Außerdem sollten Sie sich darüber im klaren sein, das Regeln für den Gebrauch von Speicheradressen und Registern eingehalten werden müssen. Eckige Klammern [] zeigen an, daß der eingeschlossene Wert eine relative Offsetadresse darstellt. Das ist ein Schlüsselelement der Speicheradressierung: Ohne Klammern wird der im Register selbst enthaltene Wert verwendet, die Zeigerfunktion also ignoriert. Einige Beispiele sollen das verdeutlichen:

`ADD AX,BX`	Der Inhalt von BX wird zu AX hinzuaddiert. Keine Speicheradressierung.
`ADD AX,[BX]`	Indirekte Adressierung: Addiert einen Wert vom Speicher zu AX hinzu; BX bestimmt den relativen Offset der Adresse des Wertes.
`ADD [BX],AX`	Der Inhalt von AX wird zu einer Speicherstelle addiert; dabei legt BX die relative Offsetadresse dieser Speicherzelle fest.
`ADD AX,123`	Unmittelbare Adressierung: Addiert den Wert 123 zu dem Wert in AX
`ADD AX,[123]`	Der Wert, der in der Speicherstelle mit dem relativen Offset 123 steht, wird zu dem Wert in AX hinzuaddiert.
`ADD AX,[BX + SI + 123]`	Indiziert-indirekte Adressierung: Der Wert, der in der Speicherstelle [BX + SI +123] steht, wird zu dem Wert in AX addiert.

2.2.2.7 Regeln zur Benutzung der Register

Sie sollten wissen, daß verschiedene Regeln zum Gebrauch der Register gehören und das die Beachtung dieser Regeln für das Erstellen von Schnittstellenroutinen auf Assemblerebene von enormer Bedeutung ist. Die Regeln und Konventionen variieren je nach den Umständen und der verwendeten Programmiersprache, so daß keine genauen Richtlinien im Sinne von Arbeitsanweisungen gegeben werden können. In diesem Abschnitt finden Sie einige generelle Regeln zusammengestellt, die sich in den meisten Fällen anwenden lassen. Weitere Hinweise zu diesem Thema finden Sie in den Kapiteln 8 bis 20. Beachten Sie bitte, daß die Regeln keinen Anspruch auf Vollständigkeit stellen.
Grundsätzlich stehen uns drei Wege offen, wie wir die Register in Schnittstellenroutinen auf Assemblerebene verwenden können. Einige Register dürfen beliebig verändert werden, anderen sollten auf ihren alten Wert zurückgesetzt werden, bevor in das aufrufende Programm zurückgesprungen wird, und der Rest sollte auf keinen Fall verändert werden.
Die Datenregister (AX bis DX) können im allgemeinen frei genutzt werden. Beachten Sie, daß das AX-Register üblicherweise für die Speicherung der Resultate verwendet wird und unter gewissen Umständen Parameter dieses Register durchlaufen, die bei Veränderungen verloren gehen.
Besondere Regeln gelten für das Arbeiten mit den vier Segmentregistern (AS, BS, SS und ES). Das CS-Register sollte niemals direkt verändert werden, obwohl es durch den Aufruf von Unterroutinen durchaus seinen Wert ändern kann. Das DS-Register kann zwar geändert, sollte aber stets wieder auf seinen Anfangswert zurückgesetzt werden. Der Ursprungswert des SS-Registers sollte im Falle einer Veränderung aufbewahrt werden. Zumeist verwenden Unterroutinen weiterhin den Stapel, auf den das SS-Register zeigt. Wird aber im Rahmen eines Programmes ein eigener Stapel geschaffen, sollte die Routine den Stand dieses Registers nach seiner Abarbeitung wiederherstellen. Das Verändern von SS kann sich auf den Basiszeiger (BP) störend auswirken. Das ES-Register darf fast immer beliebige Werte annehmen.
Der Befehlszeiger (IP oder PC) sollte ebenso wie CS nicht direkt manipuliert werden, indirekte Änderungen erfolgen automatisch und korrekt.
Der Stapelzeiger (SP) kann zwar geändert werden, aber normalerweise sind seine Veränderungen das indirekte Ergebnis der Stapelbenutzung. Den Stapel säubern entspräche einem Rücksetzen (*Reset*) des SP und ist ein wichtiger Teil der Schnittstellenkonventionen zur Benutzung von Unterprogrammen; Die Regeln für diesen Spezialfall finden Sie in den Beispielen der Kapitel 8 bis 12.
Der Basiszeiger (BP) wird in Programmen oftmals für den Zugriff auf Parameter geändert; er sollte stets wieder zurückgesetzt werden.
Die Indexregister (SI und DI) können gefahrlos manipuliert werden.

Die Statusflaggen des Flaggenregisters können durch Programme geändert werden. Manche der Flaggen zeigen bestimmte Ergebniskonditionen an, so daß ihr Setzen programmtechnisch genutzt werden kann. CF und ZF werden oft zu diesem Zweck eingesetzt. Die Kontrollflaggen und die Interruptflaggen (IF) sollten gesetzt bleiben. Es ist auch ratsam, die Richtungsflagge (DF) unangetastet zu lassen. Das Verändern der Einzelschrittflagge (TF) ist "tödlich".

2.3 Portbenutzung des 8088

Der Mikroprozessor 8088 steht mit vielen Teilen des Computers, die er steuert, in Verbindung. Er kommuniziert mit den verschiedenen ICs über die Ein-/Ausgabe-Ports. Die E/A-Ports stellen praktisch die "Türen" für den Datentransport der E/A-Geräte, wie z.B. der Tastatur oder des Drukkers, dar. Die meisten der Support-Bausteine, die in Kapitel 1 besprochen wurden, benutzen die E/A-Ports; wobei die Mehrzahl der Bausteine mehrere Portadressen für verschiedenartige Aufgaben ansteuern.
Jeder Port wird durch eine 16-bit-Portadresse zwischen 0 und 65.535 spezifiziert. Die CPU sendet Daten und Kontrollinformationen mit Hilfe der Portadressen an bestimmte Ports und der Port anwortet der CPU mit Daten oder Statusinformationen.
Ähnlich wie beim Datenzugriff verwendet der Prozessor die Daten- und Adreßbusse als Durchgänge zu den Ports. Der Zugriff auf einen Port erfolgt in zwei Schritten: Zunächst sendet die CPU ein Portkontrollsignal, anschließend wird die Portadresse auf den 16 niederwertigen Leitungen des 20-bit-Adreßbusses abgeschickt; das Gerät mit der entspechenden Portnummer reagiert.
Die Portnummern adressieren einen Speicherplatz, der Teil des E/A-Gerätes, aber nicht Teil des Hauptspeichers ist. Dem Programmierer stehen spezielle Ein- und Ausgabebefehle für Portzugriffe und die Übertragung der Daten von und zu E/A-Geräten zur Verfügung. Es gibt Geräteeinheiten, z.B. der Bildschirm-Controller, die neben ihren Portadressen auch Hauptspeicherplatz belegen. Vom Gesichtspunkt der CPU aus sind die Hauptspeicheradressen ein Teil des normalen RAM-Speichers. Dieses Konzept wird *speichereingebundene Ein-/Ausgabe* oder englisch *Memory-Mapped Input/Output* genannt. Grundsätzlich sind *Memory-Mapped*-Geräte einfacher zu programmieren als die anderen E/A-Geräte, da sie die flexiblen Speicherbefehle im Gegensatz zu den eingeschränkten Ein-/Ausgabe-Befehlen akzeptieren.

Hinweis: Der Befehlssatz des 8088 enthält die Befehle IN und OUT, um Daten von einem Port zu lesen oder in einen Port zu schreiben. BASIC hat analog die Befehle INP und OUT, die es ermöglichen, durch einfache BASIC-Programme mit verschiedenen Ports zu experimentieren und die Ports anzusprechen, ohne auf die Ebene der Assemblersprache gehen zu müssen.

2.4 Unterschiede in der Portbenutzung

Die Benutzung der verschiedenen Ports wird durch den Hardwareentwurf festgelegt. Programme, die Ports verwenden, müssen Informationen über die verschiedenen Aufgaben und Adressen der Ports haben. Größtenteils ist die Verwendung der Ports bei der gesamten PC-Modellreihe ähnlich, nur beim AT gibt es einige Abweichungen.
Bevor Sie die Portadressen in Ihren Programmen verwenden, sollten Sie die Beschreibung der ICs in Kapitel 1 durcharbeiten. In Kapitel 7 wird gezeigt, wie die Ports zur Tonerzeugung und zur direkten Hardwareprogrammierung (zwecks Kontrolle der Tonausgabe) genutzt werden können.
Das Schreiben in einen Port kann die Computerfunktionen durcheinander oder gar das System zum "abstürzen" bringen, das Lesen eines Ports ist nicht weniger gefährlich. Man kann auch nicht sagen, daß Portzugriffe, die auf dem einem PC-Modell funktionieren, auf einem anderen Modell ebenfalls zum gleichen Ergebnis führen. Das folgende Programm, das das BASIC-Kommando INP verwendet, um Daten aus einem Port zu lesen, arbeitet jedoch auf den meisten PC-Modellen einwandfrei:

```
10 FOR I = 50 TO 75
20   IF I = 64 THEN PRINT "Was passiert jetzt?"
30   PRINT I, INP (I)
40 NEXT I
```

2.5 Verarbeitung von Interrupts

An die CPU wird ein Signal, auch Interrupt genannt, gesendet, wenn ein Programm oder Gerät während der Ausführung die Hilfe des Mikroprozessors benötigt. Wenn der Prozessor das Signal erhält, unterbricht er alle anderen Aktivitäten und ruft eine im Speicher befindliche *Interruptbearbeitungsroutine* auf, die dem empfangenen Interrupt entspricht. Nachdem die Routine die erwartete Aufgabe erledigt hat, fährt das Hauptprogramm an der Stelle, an der die Verarbeitung unterbrochen wurde, fort.
Man kann die Interrupts in drei Kategorien unterteilen. Erstens gibt es bausteingenerierte Interrupts, die die Folge einer Benutzeraktion, z.B. dem Drücken einer Taste, sind. Diese Interrupts laufen über den Interrupt-Controller-Chip 8259, der die Priorität der Abarbeitung bestimmt, bevor er die Interrupts an die CPU weiterleitet. Es gibt als zweite Gruppe Interrupts, die von der CPU als Nebenprodukt auf ein außergewöhnliches Ereignis im Programmablauf ausgesendet werden, wie das beispielsweise bei der Division durch Null geschieht. Drittens existieren Interrupts, die von Programmen ausgelöst werden, um RAM- oder ROM-Unterprogramme aufzurufen. Diese Interrupts werden oft als *Softwareinterrupts* bezeichnet und sind gewöhnlich ein Teil des ROM-BIOS oder des DOS. Eine eingehende Beschreibung dieser Interrupts erfolgt in den Kapiteln 8 bis 18.

Beschreibung	Adreßbereich	
	PC/XT	AT
DMA-Controller (8237A-5)	000–00F	000–01F
Interrupt-Controller (8259A)	020–021	020–03F
Zeitgeber (8253-5; 8254.2 im AT)	040–043	040–05F
PPI (8255A-5)	060–063	entfällt
Tastatur (8042)	entfällt	060–06F
DMA-Seitenregister (74LS612)	080–083	080–09F
NMI Maskenregister (nicht-maskierbarer Interrupt)	OA	070–07F
Interrupt-Controller 2 (8259A)	entfällt	0A0–0BF
DMA-Controller 2 (8237A-5)	entfällt	0C0–0DF
Reset Arithmetik-Coprozessor	entfällt	0F0–0F1
Arithmetik-Coprozessor	entfällt	0F8–0FF
Joystick (Game-Controller)	200–20F	200–207
Erweiterungseinheit	210–217	entfällt
Anschluß für zweiten Paralleldrucker	entfällt	278–27F
erster serieller Port	3F8–3FF	3F8–3FF
zweiter serieller Port	2F8–2FF	2F8–2FF
Prototypkarte	300–31F	300–31F
Festplattenlaufwerk	320–32F	1F0–1F8
Anschluß für ersten Paralleldrucker	378–37F	378–37F
SDLC-Kommunikation (beim AT ist es der zweite Anschluß für bisynchrone Kommunikation)	380–38F	380–38F
Erster Anschluß für bisynchrone Kommunikation	entfällt	3A0–3AF
Monochromadapter/Drucker	3B0–3BF	3B0–3BF
Farb-/Grafikadapter	3D0–3DF	3D0–3DF
Laufwerks-Controller	3F0–3F7	3F0–3F7

Tabelle 2-5 Die Ports und deren Adressen, wie sie im PC/XT und AT verwendet werden

Es ist möglich, die Interruptbehandlung zu verändern oder zu erweitern, falls das für einen speziellen Verwendungszweck nötig oder wünschenswert ist.
Es gibt noch einen Sondertyp des Interrupts, den sogenannten *nicht maskierbaren Interrupts* (NMI), der sofort die Aufmerksamkeit der CPU auf sich zieht. Er wird oft zur Erkennung einer Notfallsituation, etwa einer Änderung der Versorgungsspannung oder eines Speicherfehler, verwendet. Der NMI besitzt die höchste Priorität, die CPU bearbeitet ihn vor allen anderen Interrupts.
Der "Urheber" eines Interrupts braucht die Adresse der zugehörigen Interruptroutine nicht zu kennen, sondern lediglich die Nummer des Interrupts. Die Interruptnummer zeigt auf eine Tabelle, die im unteren Speicherbereich abgelegt ist und die segmentierten Adressen der Interruptbearbeitungsroutinen enthält. Die Adresse einer solchen Routine wird *Interruptvektor* genannt und analog dazu ergibt sich der Name *Interruptvektortabelle*. Die Vektortabelle wird normalerweise vom BIOS oder DOS überwacht. Erläuterungen hierzu finden Sie in Kapitel 3. Die Erstellung

neuer Interruptunterprogramme kann sich auf bereits existierende Interruptnummern stützen, Sie können aber auch völlig neue Interrupts hinzufügen.
Die Interrupts sichern den momentanen Programmspeicher (CS) und Befehlszeiger (IP) und legen sie im Stapel ab. Dadurch kann das Programm nach der Interruptbearbeitung wieder an der ursprünglichen Stelle fortfahren. Die Flaggenregister werden ebenfalls auf den Stapel gesichert. Die Interruptflagge (IF) wird gelöscht, wodurch die Annahme weiterer Interrupts vorläufig unmöglich wird. Die Interruptroutinen setzen IF aber so bald als möglich wieder zurück, meist schon am Anfang der Routine. Dafür gibt es einen speziellen Interruptbefehl, IRET, der dem RET-Befehl bei Unterprogrammen entspricht. IRET bringt die Flaggen und die Register CS und IP auf den ursprünglichen Stand zurück.
Es ist nicht ungewöhnlich, daß Assemblerroutinen an Programme oder sogar Programmiersprachen angehängt werden. Das ermöglicht den Zugriff auf DOS- und BIOS-Unterroutinen, wodurch die Leistungsfähigkeit der Programme, die in dieser Programmiersprache geschrieben sind, sehr wirksam gesteigert werden kann. Schnittstellenroutinen, speziell zum Aufrufen von DOS- oder BIOS-Dienstleistungen, müssen in Assembler programmiert werden. In den meisten Fällen bestehen diese Programme aus einfachen Unterprogramm- und Interruptaufrufen mit dem Befehl INT. Nur sehr anspruchsvolle Assemblerprogrammierung erfordert das Erstellen neuer Interruptbearbeitungsroutinen und den Befehl IRET. Beispiele und Erklärungen zur Assemblerprogrammierung finden Sie in den Kapiteln 8 bis 20.

2.6 Stapel

Der Stapel ist ein fester Bestandteil der Konzeption des 8088 Prozessors. Mit dem Stapel steht den Programmen ein fester Speicherbereich zur Verfügung, in dem Daten abgelegt werden können und der vom Prozessor verwaltet wird. Im Stapel wird beispielsweise gespeichert, von wo Unterprogramme aufgerufen wurden und welche Parameter übergeben wurden. Der Stapel kann auch als temporärer Arbeitsspeicher verwendet werden, was jedoch kaum von Bedeutung ist.
Der Stapel funktioniert, wie der Name schon sagt, nach einem Stapelprinzip. Daten werden am oberen Ende hineingegeben und auch von oben wieder weggenommen. Ein Stapel arbeitet immer nach dem Motto: "Wer zuletzt herein gekommen ist, geht zuerst wieder heraus" (*Last-In-First-Out* oder LIFO). Das bedeutet beispielsweise, daß zu dem als letztes Unterprogramm aufgerufenen Programm als erstes wieder zurückgesprungen wird. Auf diese Art speichert der Stapel die Reihenfolge der Abarbeitung von Programmen, Unterprogrammen und Interruptroutinen, gleichgültig, wie komplex die Verschachtelung ist.

Ein Stapel wird von unten (höchste Adresse) nach oben (niedrigste Adresse) verwendet. Wenn Daten in den Stapel geschrieben werden, kommen sie in den Speicherplatz direkt unterhalb des momentan "oberen Endes" des Stapel. Der Stapel wächst im Speicher nach unten, das heißt, wenn Daten hinzugefügt werden, wird das "obere Ende" des Stapels zu niedrigeren Speicherstellen wandern, wobei jedesmal SP vermindert wird. Es ist wichtig, daß Sie das Stapelprinzip verstehen, da es recht häufig zur Anwendung kommt.

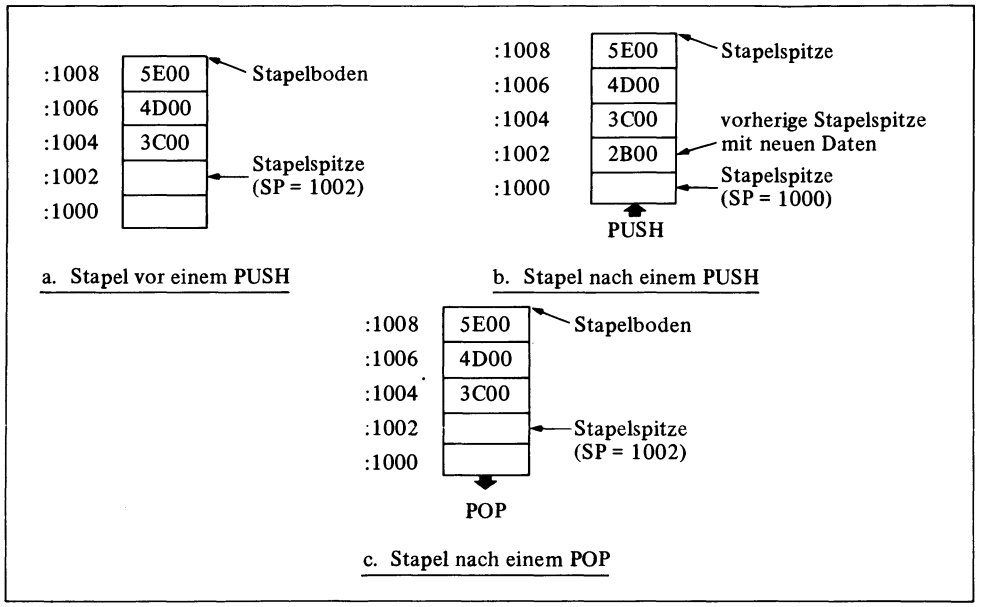


Bild 2-5 Beispiel für die Benutzung des Stapels

Jeder Programmteil kann zu jedem Zeitpunkt einen neuen Stapelspeicherbereich anlegen, was aber für gewöhnlich nicht getan wird. Vielmehr wird im allgemeinen mit einem Programm auch ein einziger Stapel eingerichtet und während des gesamten Programmablaufes beibehalten. Sowohl das Programm als auch die Unterprogramme und DOS- und BIOS-Routinen, die aufgerufen werden, verwenden diesen einen Stapel. Wenn gerade kein Programm ausgeführt wird, benutzt DOS einen eigenen Stapel.
Leider gibt es keine Faustregel für das Berechnen der Stapelgröße, die ein Programm benötigt. Der 8088 stellt auch keine Möglichkeit bereit, wie man leicht erkennen könnte, daß der Stapelspeicherplatz knapp wird. Das kann einen Programmierer schon nervös machen und die Wahl der Größe

des Stapelspeichers sehr erschweren. Die meisten Programmiersprachen, einschließlich interpretierendem BASIC, verwenden automatische einen Stapel von 512 Bytes Länge, solange nichts anderes spezifiziert wird. Sie können davon ausgehen, daß 512 Bytes für den Stapel ausreichend sind, wenn Sie keine speziellen Gründe haben, etwas anderes anzunehmen.

2.7 Umgekehrtes Speichern von Worten

Während der Speicher des Computers in Einheiten zu je acht Bits, einem Byte, organisiert ist, gibt es eine ganze Reihe von Funktionen, die ein 16-bit-Wort verlangen. Diese Worte werden in zwei aufeinanderfolgenden Speicherstellen (zu je einem Byte) abgelegt. Das LSB (*Least-Signifikant Byte*) kommt in die niedrigere Speicherstelle, das MSB (*Most-Significant Byte*) in die höhere Speicherstelle. Je nach Gesichtspunkt erscheint diese Reihenfolge anders, als man es vielleicht erwartet. Wegen dieser Erscheinung nennt man dieses Speicherkonzept zuweilen auch *umgekehrtes Speichern eines Wortes.*

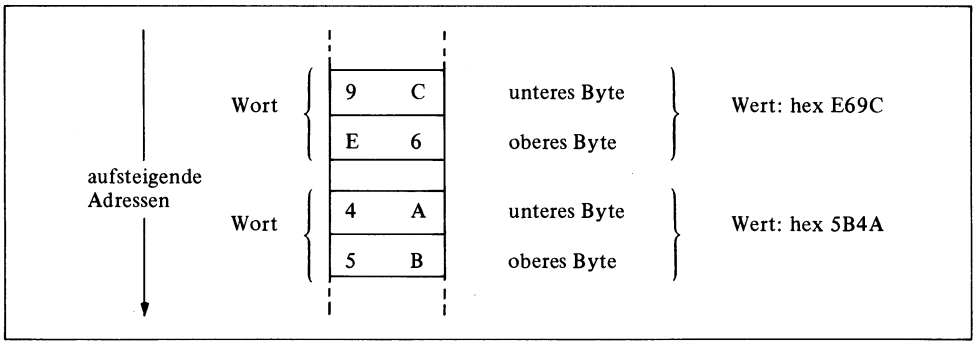


Bild 2-6 Umgekehrtes Speichern von Worten

Wenn Sie mit Bytes und Worten im Speicher arbeiten, sollten Sie sich nicht durch die umgekehrte Speicherung durcheinanderbringen lassen. Die Quelle der Verwirrung liegt meist darin, wie wir unsere Daten zu Papier bringen. Schreiben wir z.B. ein Wort in hexadezimaler Notation, kann das folgendermaßen aussehen: ABCD. Die Reihenfolge der Wertigkeit ist dieselbe, wie in dezimaler Schreibweise: Die höchstwertige Stelle steht am Anfang (links). Schrieben wir unser Beispiel aber so, wie es abgespeichert wird, dann käme eben der Inhalt der niedrigeren Speicherstelle zuerst: CDAB. In anderen Worten, die Hexzahl ABCD wird im Speicher als CD in der niederen und AB in der höheren Adresse abgelegt.

Kapitel 3
ROM-Software

3.1 ROM-Startprogramm 42
3.1.1 DOS-Grundlagen 44
3.2 ROM-BIOS 45
3.2.1 Interruptvektoren 46
3.2.2 Speicherstellen im unteren Speicherbereich 51
3.2.2.1 Speicherbereiche für interne Kommunikation 58
3.2.3 ROM-Versionen und Modellidentifikation 58
3.3 ROM-BASIC 61
3.4 ROM-Erweiterungen 62
3.5 Anmerkungen 63

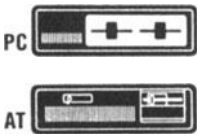
Um einen Computer zum Arbeiten zu bewegen, braucht man Software. Sind Teile dieser Software schon fest eingebaut, so vereinfacht das natürlich vieles. Hier setzen die Aufgaben des sog. *ROM* ein.
ROM steht für den englischen Begriff *Read-Only Memory*, zu deutsch *Nur-Lese-Speicher*. Der Inhalt des ROM kann nicht verändert werden oder verloren gehen. Die PCs werden mit einer Minimalausstattung an ROM geliefert, die einen Systemstart und das Arbeiten mit der CPU und den Peripheriegeräten erlaubt. Der große Vorteil dieser im ROM gespeicherten Basisprogramme liegt darin, daß sie nicht wie DOS von einer Diskette geladen werden müssen, sondern immer bereit stehen. Deshalb basieren die meisten Programme, auch das Betriebssystem DOS, auf der ROM-Basissoftware.
Vier unterschiedliche Basisprogramme sind im ROM der PCs abgelegt: die Systemstartprogramme, die den Computer zur Arbeitsaufnahme konfigurieren (auch *Start-up* genannt); das ROM-BIOS (*Basic Input/Output System*, grundlegendes Ein-/Ausgabesystem), das eine Reihe von *Dienstleistungen* (sog. *Services*) auf Maschinenspracheebene bereitstellt, die für den laufenden Betrieb des Computers nötig sind; das ROM-BASIC, das den Kern der Programmiersprache BASIC beinhaltet; und die ROM-Erweiterungen, auf die zurückgegriffen werden kann, wenn weitere Geräte an den Computer angeschlossen werden. In diesem Kapitel beschäftigen wir uns ausführlich mit dem ROM und seinem Umfang.
Der Speicherblock mit den höchsten Adressen, der für ROM-Programme reserviert wird, fängt mit der Segmentadresse hex F000 an. Die PC-Modelle benutzen von diesem Speicherbereich, der 64 KB umfaßt, unterschiedlich große Teile, je nach der Komplexität der jeweiligen ROM-Software. Der Original-PC ist ein relativ einfaches Gerät, er braucht nur 40 Kbyte der zur Verfügung stehenden 64 Kbyte, während der AT den vollen Speicherplatz aufgrund seiner komplexen Hardwarestruktur nutzt.

3.1 ROM-Startprogramm

Die erste Aufgabe des ROM besteht darin, den Systemstart des Computers durchzuführen. Mit der Programmierung hat diese Startsoftware sehr wenig zu tun, es ist aber trotzdem nützlich, sie wenigstens in den Grundlagen zu verstehen.
Den Startprogrammen fallen mehrere Pflichten zu. Sie führen einen Zuverlässigkeitstest des Computers und der ROM-Software durch und stellen dadurch sicher, daß alles einwandfrei funktioniert. Sie initialisieren die Bausteine und die Standardausrüstung des Computers, legen die Interruptvektortabelle an und überprüfen, ob optionale Geräte angeschlossen sind. Ist eine Diskettenstation vorhanden, so besteht die letzte Aufgabe des ROM darin, das Laden des Betriebssystems von einer Diskette zu veranlassen.

Der Zuverlässigkeitstest gehört zu einem Testteil, der als *Einschalt-Selbsttest* oder *Power On Self Test* (POST) bekannt ist. Dies ist ein wichtiger erster Schritt nach dem Einschalten des Gerätes, denn immerhin gilt es, herauszufinden, ob der Computer überhaupt einsatzbereit ist. Die POST-Routinen sind alle sehr kurz. Nur die Speichertests, die sich, je nach Speicherkapazität, in die Länge ziehen können, verlängern den Einschaltselbsttest.

Der Initialisierungsprozeß ist etwas komplizierter. Ein Unterprogramm setzt die Interruptvektoren auf ihre Anfangswerte. Diese Werte zeigen entweder auf die Standard-Interruptroutinen im ROM-BIOS, oder auf noch "leere" Routinen, die unser Programm später bereitstellen wird. Eine andere Initialisierungsroutine stellt fest, mit welchen Geräteeinheiten und mit welcher Ausstattung der Computer versehen ist und schreibt diese Informationen sodann in die dafür vorgesehenen Standardspeicherstellen im unteren Speicherbereich. Die Hardwareausstattung beschäftigt uns noch später in diesem Kapitel. Wie diese Informationen erhalten werden, variiert von Modell zu Modell. Im Standard-PC wird sie der Stellung zweier Schalterreihen entnommen, die auf der Systemplatine des Computers vorhanden sind. Im AT werden diese Angaben aus einem speziellen nichtflüchtigen Speicherbereich, der von den Diagnoseprogrammen gesetzt wird, gelesen.



Die Überprüfungsergebnisse über die angeschlossenen Geräteeinheiten und Erweiterungskarten (der Ausstattungsstatus) werden bei allen PC-Modellen auf identische Weise intern festgehalten. Software, die den Gerätestatus feststellen will, hat also nur eine Abfrage speziell zu berücksichtigen. Die Initialisierungsroutinen suchen auch nach neuer Ausrüstung und ROM-Erweiterungen. Wird eine solche gefunden, wird die Kontrolle sofort an diese ROM-Erweiterungen übergeben, damit sie sich selbst initialisieren können. Dann fahren die Initialisierungsprogramme mit der Ausführung der verbleibenden Startroutinen fort (mehr hierzu später).

Der letzte Teil der Startprozedur, nach dem POST-Test, dem Initialisierungsprozeß und dem Einfügen der ROM-Erweiterungen, wird *Boot-Strap-Loader* oder *Urlader* genannt. Dies ist ein kurzes Programm, das das Laden eines Programmes von Diskette veranlaßt. Kurz gesagt besteht die Aufgabe des ROM-Urladers im Suchen eines Eintrages, der *Boot-Eintrag* genannt wird, und sich, falls überhaupt vorhanden, an einer bestimmten Stelle auf einer Diskette befindet. Wird ein solcher Eintrag gefunden, wird die Kontrolle vom Computer auf dieses Programm übertragen. Das Kurzprogramm, das im Boot-Eintrag enthalten ist, kann ein anderes wesentliche längeres Programm von der Diskette laden, falls dies gewünscht wird. Dieses ist gewöhnlich ein Diskettenbetriebssystem, wie DOS, kann aber auch ein anwendungsbezogenes und selbstladendes Programm sein, wie z.B. Microsofts "Flight Simulator".

Findet der ROM-Urlader keinen Disketteneintrag, aktiviert er das eingebaute ROM-Kassetten-BASIC oder meldet, daß nicht gebootet werden kann. Sobald eine dieser Prozeduren beendet wurde, ist das "Hochfahren" (Starten) des Computers beendet und die Benutzerprogramme können ihre Arbeit aufnehmen.

Hinweis: Die ROM-Erweiterungen können den Boot-Prozeß verändern oder sogar verhindern.

3.1.1 DOS-Grundlagen

Nachdem der ROM-Urlader die Kontrolle an den Boot-Eintrag einer Diskette übergeben hat, sucht dieser auf der Diskette nach DOS. Dies erfolgt, indem zwei versteckte Dateien, IBMBIO.COM und IBMDOS.COM, gesucht werden und diese, falls gefunden, zusammen mit dem DOS-Befehlsinterpreter, COMMAND.COM, in den Speicher geladen werden. Während des Ladeprozesses können auch optionale Teile des DOS, wie z.B. installierbare Schnittstellentreiber, mitgeladen werden.

Die IBMBIO.COM-Datei enthält Erweiterungen des ROM-BIOS. Diese Erweiterungen können Veränderungen der normalen Ein-/Ausgabe-Operationen bewirken oder neue E/A-Funktionen hinzufügen; oftmals enthält IBMBIO.COM auch Korrekturen des existierenden ROM-BIOS, neue Routinen für spezielle Geräte oder benutzerbezogene Änderungen der Standard-ROM-BIOS-Routinen. Da alle diese Erweiterungen und Veränderungen ein Teil der Diskettensoftware sind, müssen sie geeignete Mittel zur Modifikation des ROM-BIOS mit sich bringen. Dabei müssen nicht nur die neuen Routinen eingefügt, sondern auch die Interruptvektoren entsprechend geändert werden. Immer wenn neue Geräte angeschlossen werden, kann das zugehörige Unterstützungsprogramm in die IBMBIO.COM-Datei oder als installierbarer Schnittstellentreiber aufgenommen werden. Das befreit uns von dem ansonsten notwendigen Auswechseln der ROM-Bausteine. Näheres über Schnittstellentreiber finden Sie in Anhang A.

Wir können die ROM-BIOS-Routinen als niedrigste, uns zugängliche Softwareebene ansehen, die die fundamentalen und recht "primitiven" Ein-/Ausgabe-Operationen durchführt. Die IBMBIO.COM-Routinen befinden sich auf fast derselben Stufe und führen Grundoperationen durch. Bei einem Vergleich müssen die IBMDOS.COM-Routinen jedoch als ein klein wenig höher entwickelt eingestuft werden. Sie stehen aber natürlich noch unter den Programmiersprachen, die in diesem Kontext die oberste Ebene bilden.

Die IBMDOS.COM-Datei enthält die DOS-Routinen. Diese werden, wie die BIOS-Routinen, durch Interrupts aufgerufen, deren Vektoren in einer Tabelle im unteren Speicherbereich angesiedelt sind. Einer der DOS-Interrupts, Interrupt 33 (hex 21), ist besonders wichtig. Wenn er aufgerufen wird, gibt er uns Zugang zu Sekundärroutinen, die *DOS-Funk-*

tionen genannt werden. Diese DOS-Funktionen ermöglichen uns eine bessere und effizientere Kontrolle der Ein-/Ausgabe-Operationen, als dies mit den BIOS-Routinen möglich ist. Das gilt vor allem für Diskettenoperationen. Alle Standard-Diskettenprozesse können über das DOS ausgeführt werden: Diskettenformatierung, Lesen und Schreiben von Daten, Öffnen, Löschen und Schließen von Dateien und das Durchsuchen des Verzeichnisses. Die "höheren" DOS-Programmteile, wie FORMAT, COPY und DIR, verwenden ebenfalls zumeist die DOS-Routinen. Wir können diese DOS-Funktionen auch in unsere Programme einbauen, wenn wir eine detailliertere Kontrolle über bestimmte Vorgänge benötigen, und unsere Programmiersprache diese nicht bereitstellt, wir aber andererseits zögern, auf die BIOS-Routinen zuzugreifen. Die DOS-Routinen sind ein wichtiger Bestandteil dieses Buches und wir haben ihnen daher fünf Kapitel (Kapitel 14 bis 18) gewidmet.

Die COMMAND.COM-Datei ist der dritte und, zumindest vom Standpunkt des Benutzers aus, wichtigste Teil des DOS. Die Datei enthält die Routinen, die die über die Tastatur eingegebenen Befehle interpretieren, wenn wir im DOS-Befehlsmodus arbeiten. Aufgrund des Vergleichs der Eingabe mit Tabelleneinträgen der Befehlsnamen kann das COMMAND.COM-Programm zwischen internen Befehlen, die ein Teil des COMMAND.COM sind, wie ERASE oder RENAME, und externen Befehlen der DOS-Benutzerprogramme (DEBUG, EDLIN) oder einem unserer eigenen Befehle unterscheiden. COMMAND.COM beantwortet unsere Befehle, indem es die Ausführung der entsprechenden Routinen veranlaßt, wenn Befehle sich auf eine COMMAND.COM-interne Routine beziehen, oder andernfalls auf der Diskette nach dem verlangten Programm sucht und es in den Speicher lädt. Die ganze COMMAND.COM Datei ist sehr wichtig und ich empfehle Ihnen, zur weiteren Information die entsprechenden Abschnitte im DOS-Handbuch zu lesen.

3.2 ROM-BIOS

Das ROM-BIOS ist ein Teil des ROM-Speichers, der die ganze Zeit über aktiv ist, während der Computer arbeitet. Das ROM-BIOS stellt eine ganze Reihe unterschiedlicher Routinen zur Verfügung, die für die Basisperationen des Computers benötigt werden. Die Peripherieüberwachung wird zum größten Teil vom BIOS übernommen, wobei zur Peripherie auch der Bildschirm, die Tastatur, die Disketten- und Plattenstation gehören. Wenn wir den Begriff BIOS (*Basic Input/Output System*) im engen Sinne verwenden, so sind damit nur die Ein-/Ausgaberoutinen gemeint, die einen einfachen Befehl wie z.B. "Lies etwas von der Diskette" in die einzelnen Schritte zerlegen, die unternommen werden müssen, um die Anweisung auszuführen. Dazu gehören auch die Fehlererkennung und die Fehlerkorrektur, soweit das möglich ist. Im weitesten Sinne kon-

trolliert das BIOS nicht nur Ein-/Ausgabefunktionen, sondern enthält auch Routinen zur Überwachung anderer elementarer Funktionen, wie z.B. zur Kontrolle der aktuellen Tageszeit.
Die Routinen des BIOS sind Softwareprogramme, die zwischen unseren Programmen und der Hardware vermitteln. Das BIOS arbeitet dabei in beide Richtungen, es hat sozusagen "zwei Gesichter", eines nach oben (zur Software hin), das andere nach unten (zur Hardware hin). Die "obere" Seite erhält die Anforderungen der Programme, die Standard-BIOS-E/A-Dienstleistungen abzuarbeiten. Diese Anforderungen bestehen aus der Kombination einer Interruptnummer (die die allgemeine Dienstleistung wie z.B. Drucken spezifiziert) und einer Programmroutinennummer, die die speziell gewünschte Routine anwählt. Auf der anderen ("unteren") Seite kommuniziert BIOS mit den Hardwaregeräten (z.B. Bildschirm oder Diskettenstation) und verwendet hierbei genau die Befehle, die das jeweilige Gerät benötigt. Diese Seite des BIOS bearbeitet auch die von den Geräten kommenden Hardwareinterrupts. Bei jedem Tastendruck z.B. erzeugt die Tastatur einen Hardwareinterrupt, der dem BIOS diese Handlung (den Tastendruck) mitteilt.
Von der gesamten ROM-Software sind die BIOS-Dienstleistungen für den Programmierer von großem Interesse. In den Kapitel 8 bis 13 werden sie deshalb ausführlich behandelt, so daß wir an dieser Stelle nicht näher darauf eingehen wollen. Beschäftigen werden wir uns aber nun mit den Einsatzmöglichkeiten und den Kontrollfunktionen des BIOS, die die Ein-/Ausgabe-Prozesse des Computers steuern.

3.2.1 Interruptvektoren

Der IBM PC wird, wie alle Computer, die auf der Intel 8086-CPU basieren, weitgehend durch die Interrupts, die sowohl von der Software als auch von der Hardware generiert werden können, gesteuert. Die BIOS-Routinen machen hier keine Ausnahme. Jede hat eine ihr zugeordnete Interruptnummer, die aufgerufen werden muß, soll der entsprechende Service ausgeführt werden.
Wird ein Interrupt erzeugt, wird die Kontrolle von der CPU an das zugehörige Interruptbearbeitungsprogramm abgegeben, das zumeist im ROM gespeichert ist (eine BIOS-Routine ist in der Regel ein Programm, das eine Programmunterbrechung abarbeitet). Die Bearbeitungsroutine wird aufgerufen, indem ihre segmentierte Adresse in die Register geladen wird, die für den Programmablauf verantwortlich sind: das Codesegmentregister CS (*Code Segment*) und das Befehlszeigerregister IP (*Instruction Pointer*), auch als Registerpaar CS:IP bekannt. Die segmentierten Adressen werden in diesem Zusammenhang *Interruptvektoren* genannt.
Die Interruptvektoren werden beim Systemstart gesetzt und zeigen auf die Interruptbearbeitungsroutinen im ROM. Sie werden im RAM als Wort-

paare gespeichert. Der erste Teil umfaßt die relative Offsetadresse, der zweite die Segmentadresse (der 8088 speichert sie in umgekehrter Reihenfolge; siehe auch Kapitel 2.7. Sie finden dort eine Erklärung dieses umgekehrten Speicherformats). Die Interruptvektoren können durch einfaches Verändern der Speicherstelle, in denen sie liegen, nach Wunsch auf eine beliebige neue Interruptroutine gesetzt werden.
Es gibt sieben Interruptarten: CPU-, Hardware-, Software-, DOS-, BASIC-, Adreß- und allgemeine Interrupts.
Die CPU-Interrupts, die auch logische Interrupts genannt werden, sind in den Mikroprozessor selbst integriert. Vier von ihnen (Interrupts 0, 1, 3 und 4) werden vom Prozessor selbst erzeugt, und einer (Interrupt 2, der nicht-maskierbare Interrupt NMI) wird durch ein Signal von außen aktiviert.
Die Hardwareinterrupts sind fest in den PC eingebaut. Acht dieser Interrupts sind mit der CPU oder der Hauptplatine fest verdrahtet. Sie können nicht verändert werden. Alle Hardwareinterrupts werden vom 8259A PIC-Baustein überwacht, die reservierten Codes sind 2, 8, 9 und von 11 bis 15.
Die Softwareinterrupts sind ein Teil der ROM-BIOS-Routinen. Diese Unterprogramme können selbst nicht verändert werden, jedoch lassen sich die Interruptvektoren manipulieren, so daß sie auf andere Routinen zeigen. Die dafür vorgesehenen Codes sind 5, 16 bis 28 (inklusive) und 72.
Die DOS-Interrupts sind bei der Benutzung des DOS immer zugänglich. Viele Programme und Programmiersprachen nutzen die vom DOS bereitgestellten Laufwerks-Ein-/Ausgaberoutinen, die mit Hilfe der DOS-Interrupts aufgerufen werden können. Die hierfür reservierten Codes liegen zwischen 32 und 255 (32 bis 96 werden benutzt, die anderen nur bereitgestellt).
Die BASIC-Interrupts werden durch BASIC selbst aufgerufen. Sie sind immer aufrufbar, solange BASIC aktiviert ist. Die hierfür verwendeten Codes sind 128 bis 240.
Die Adreßinterrupts sind Teil der Interruptvektortabelle und werden zur Speicherung segmentierter Adressen verwendet. Mit diesen Interrupts sind keine Interruptbearbeitungsroutinen verbunden und sie sind somit gar keine tatsächlichen Interrupts. Drei von ihnen hängen aber mit sehr wichtigen Verzeichnissen zusammen: die Bildschirminitialisierungstabelle, die Diskettenbasistabelle und die Grafikzeichentabelle. Diese Tabellen beinhalten Parameter, die das ROM-BIOS in der Startprozedur bzw. für das Erzeugen von Grafikzeichen benötigt. Die den Adressinterrupts zugeordneten Codes liegen zwischen 29 und 32.
Die allgemeinen Interrupts werden vom aktuellen Programm zeitweilig angelegt. Ihre Codes befinden sich zwischen 96 und 103.
Die Interruptvektoren werden im untersten Speicherbereich abgelegt. Der erste Speicherplatz umfaßt den Vektor der Interruptnummer 0, und so weiter. Da jeder Interruptvektor die Länge von 2 Worten hat, muß die

Interruptnummer mit vier multipliziert werden, um die absolute Speicheradresse des Vektors zu erhalten. So steht z.B. der Vektor des Interrupt 5 für Bildschirmausgabe an der Adresse mit dem Byte-Offset 20 (5 x 4 ergibt 20). Man kann die Interruptvektoren überprüfen, indem man diese dezimale Zahl in eine Hexzahl umwandelt und DEBUG benutzt. Der Befehl DEBUG akzeptiert nur Hexzahlen. Die dezimale Zahl 20 entspricht hex 14. Wollen Sie den Inhalt des oben schon erwähnten Interruptvektors wissen, so geben Sie folgende Befehle ein:

```
DEBUG
D 0000:0014 L 4
```

Als Ergebnis erhalten Sie die vier Bytes (hex) angezeigt:

```
54 FF 00 F0
```

In eine segmentierte Adresse umgewandelt und unter Berücksichtigung der umgekehrten Speicherung läßt sich der Interruptvektor, der auf den Anfangspunkt der Bildschirmdruckroutine im ROM deutet, als F000:FF54 erkennen. Mit dem DEBUG-Befehl kann man ohne jegliche Schwierigkeiten auch jeden anderen Interruptvektor einsehen.
Tabelle 3-1 zeigt eine Auflistung der Hauptinterrupts und ihrer Vektorpositionen. Dies sind für den Programmierer die wichtigsten Interrupts. In den Kapiteln 8 bis 18 finden Sie Details der meisten Interrupts genannt. Interrupts, die nicht in diese Liste aufgenommen wurden, dienen größtenteils der zukünftigen Weiterentwicklung der IBM PC-Architektur. Sie sind vorsorglich vorhanden, ohne derzeit eine tatsächliche Funktion zu erfüllen.

Verändern der Interruptvektoren

Von großen Interesse ist natürlich das Verändern eines Interruptvektors, so daß dieser auf eine andere als die Standard-Unterroutine zeigt. Um eine solche eigene Routine einzurichten, gehen Sie wie folgt vor. Zunächst schreiben Sie Ihre neue Routine und legen sie im RAM ab. Anschließend teilen Sie einfach einem schon existierenden Interruptvektor eine neue Zeigeradresse zu, die auf den Einsprungspunkt ihrer RAM-Routine verweist. Bedenken Sie, daß dieser Einsprungspunkt nicht notwendigerweise das erste Byte der Routine sein muß.
Die Manipulation eines Vektors kann Byte für Byte auf der Ebene der Assemblersprache stattfinden, aber auch durch einen entsprechenden Befehl in einer höheren Programmiersprache. In BASIC gibt es hierfür den Befehl POKE. In einigen Fällen besteht die Möglichkeit, daß ein Interrupt, der gerade geändert wird, just in diesem Moment zur Ausführung gelangt. Das kann zu völligem Chaos führen, falls z.B. eines der

Interrupt Dez	Interrupt Hex	Adresse	Verwendung
0	0	0000	Wird von der CPU bei Division durch Null erzeugt
1	1	0004	Dient für Einzelschrittbetrieb (wie z.B. in DEBUG)
2	2	0008	Nicht-maskierbarer Interrupt NMI
3	3	000C	Dient zum Setzen von Haltepunkten (Break Points) in Programm (z.B. in DEBUG)
4	4	0010	Wird bei arithmetischem Überlauf erzeugt
5	5	0014	Ruft BIOS-Bildschirmdruckroutine auf
8	8	0020	Wird vom Hardware-Taktimpuls erzeugt
9	9	0024	Entsteht bei Tastenanschlägen
13	D	0034	Entsteht während der Strahlrücklaufzeit der Kathodenstrahlröhre (Bildschirmsteuerung)
14	E	0038	Signalisiert Laufwerksoperation
15	F	003C	Wird für Druckersteuerung benutzt
16	10	0040	Ruft BIOS-Bildschirmausgaberoutinen auf
17	11	0044	Ruft BIOS-Ausstattungsroutine auf
18	12	0048	Ruft BIOS-Speicherroutinen auf
19	13	004C	Ruft BIOS-Laufwerksroutinen auf
20	14	0050	Ruft BIOS-Kommunikationsroutinen auf
21	15	0054	Ruft BIOS-Kassettenrecorderroutinen auf
22	16	0058	Ruft Standard-BIOS-Tastaturroutinen auf
23	17	005C	Ruft BIOS-Druckerroutinen auf
24	18	0060	Aktiviert ROM-BASIC (falls vorhanden)
25	19	0064	Ruft BIOS-Systemstartroutine auf
26	1A	0068	Ruft BIOS-Zeit/Datum-Routinen auf
27	1B	006C	Wird durch Unterbrechungstastendruck unter BIOS erzeugt; wenn eine Routine vorhanden ist, wird sie aufgerufen
28	1C	0070	Wird bei jedem Taktimpuls erzeugt; wenn eine Routine vorhanden ist, wird sie aufgerufen
29	1D	0074	Zeiger auf Bildschirmparametertabelle
30	1E	0078	Zeiger auf Laufwerksparametertabelle
31	1F	007C	Zeiger auf obere Bildschirmgrafikzeichen
32	20	0080	Ruft DOS-Programmbeendigungsroutine auf
33	21	0084	Dient als DOS-Hauptinterrupt zum Aufruf aller DOS-Funktionsaufrufe
34	22	0088	Wenn vorhanden, wird nach der DOS-Programmbeendigungsroutine zu der hier festgelegten Routine verzweigt
35	23	008C	Wenn vorhanden, wird bei einer Tastaturunterbrechung unter DOS zu der hier festgelegten Routine verzweigt
36	24	0090	Wenn vorhanden, wird bei einem gravierenden Fehler unter DOS zu der hier festgelegten Routine verzweigt
37	25	0094	Ruft DOS-Routine zum absoluten Lesen auf
38	26	0098	Ruft DOS-Routine zum absoluten Schreiben auf
39	27	009C	DOS-Routine: Programm beenden, aber im Speicher verbleiben
73	49	0124	Zeiger auf Umsetzungstabelle für tastaturgestützte Geräte

Tabelle 3-1 Die Hauptinterrupts des IBM PC

beiden Vektor-Bytes schon geändert wurde, das andere aber noch nicht. Wenn Sie diese Gefahr nicht fürchten, dürfen Sie bedenkenlos POKE verwenden. Andernfalls gibt es zwei Wege, die das Verändern eines Vektors möglich machen, ohne daß die Gefahr eines Aufrufs während der Änderung besteht. Hierzu sind allerdings spezielle Sicherheitsvorkehrungen notwendig.
Die erste Methode besteht darin, den Vektor sozusagen "per Hand" zu ändern und dabei den Interrupt zuvor quasi abzuschalten. Dies geschieht mit dem Befehl zum Abschalten von Interrupts (*Clear Interrupt* oder CLI). CLI unterdrückt alle Interrupts mit Ausnahme des nicht-maskierbaren (NMI), der eben aus dem Grund, daß er nicht abschaltbar ist, *nicht-maskierbar* heißt. Der NMI sollte daher eigentlich nur in Ausnahmefällen verwendet werden. Wegen der stets bestehenden Gefahr eines NMI bietet diese Methode zur Änderung eines Interrupts, unter Zuhilfenahme des Befehls CLI, leider keine absolute Sicherheit.
Sie sehen hier zwei Beispiele, die mit Hilfe des CLI-Befehls einen Interruptvektor verändern. Das erste Beispiel setzt den neuen Vektor mit zwei MOV-Befehlen, die die zwei Vektorwerte transferieren:

```
XOR     AX,AX              ;Segmentregister löschen
MOV     ES,AX              ;Segmentregister löschen
CLI                        ;Interrupts unterdrücken
MOV     WORD PTR ES:36,XX  ;Vektoroffsetteil transferieren
MOV     WORD PTR ES:38,YY  ;Vektorsegmentteil transferieren
STI                        ;Interrupts wieder zulassen
```

Bei dieser Methode besteht immer die Gefahr, daß zwischen zwei MOV-Befehlen ein NMI gesendet wird. Die Wahrscheinlichkeit ist zwar gering, aber nicht auszuschließen. Diese Gefahr kann allerdings verringert werden, indem beide MOV-Befehle in eine einzige Mehrfach-MOV-Anweisung (String-MOV) gepackt werden (MOVS). Dazu müssen jedoch zunächst diverse Register entsprechend gesetzt werden. Das zweite Beispiel erzeugt dasselbe Resultat wie das erste Beispiel:

```
;Einrichten diverser Register für Mehrfach-String-MOV
XOR     DI,DI      ;Nullwort erzeugen
MOV     ES,DI      ;64 Kbyte Datenblock auf 0 setzen
MOV     DI,36      ;Transfer-nach-Offset setzen (Zielstringoffset)
MOV     SI,XXXX    ;Transfer-von-Offset setzen (Quellstringoffset)
MOV     CX,2       ;Anzahl der zu verschiebenden Worte
CLD                ;Vorwärts-Richtung festlegen
;Dann Transferierung bei ausgeschalteten Interrupts
CLI                ;Interrupts unterdrücken
REP     MOVSW      ;wiederholte Verschiebung der Worte
STI                ;Interrupts wieder zulassen
```

Sie haben nun zwei Möglichkeiten kennengelernt, den Interruptvektor im "Selbsthilfeverfahren" zu verändern. Eine Alternative bietet sich mit der Dienstleistungsroutine Nummer 37 an, die von DOS bereitgestellt wird. DOS verändert den Vektorwert auf die sicherste Art und Weise. Außerdem bleibt die Kompatibilität zu neuen Betriebssystementwicklungen erhalten, was bei einer Änderung "von Hand" nicht unbedingt gegeben ist. Immerhin beginnt mit dem AT eine Entwicklung, die den Interruptvektoren und den Segmentregistern weiterreichende Aufgaben als bisher überträgt. Das folgende Beispiel zeigt, wie die DOS-Hilfsroutine 37 aufgerufen wird.

```
MOV     DX,XX       ;Offsetteil des Vektors laden
MOV     DS,YY       ;Segmentteil des Vektors laden
MOV     AH,37       ;Interrupt-setzen-Funktion anwählen
MOV     AL,9        ;Interrupt Nummer 9 ändern
INT     33          ;DOS-Funktionsaufruf-Interrupt
```

Dies ist der einfachste Weg, die DOS-Programmteile anzusprechen. In diesem Routinenaufruf liegt aber dennoch eine Schwierigkeit: Das DS-Register muß mit einer der Adressen, die an DOS übergeben wird, gefüllt werden, was den Zugriff auf normale Daten durch dieses Register behindert. Zur Umgehung dieses Problems ist ein relativ großer Umweg vonnöten. Es folgt ein Programm, das aus der Programmsammlung *Norton Utilities* stammt:

```
PUSH    DS                  ;derzeitiges Datensegment sichern
MOV     DX,OFFSET PGROUP:XXX ;Vektoroffset holen
PUSH    CS                  ;Codesegment in ...
POP     DS                  ;... das Datensegment transferieren
MOV     AH,37               ;Interrupt-setzen-Funktion wählen
MOV     AL,9                ;Interrupt 9 ändern
INT     33                  ;DOS-Funktionsaufruf-Interrupt
POP     DS                  ;Original-Datensegment wieder holen
```

3.2.2 Speicherstellen im unteren Speicherbereich

Viele Operationen des IBM PC werden durch Daten gesteuert und kontrolliert, die im unteren Adreßbereich des Computerpeichers abgelegt werden. Besonders wichtig sind die 256-byte-Bereiche sowohl ab hex 400 als auch ab hex 500. Die Daten werden vom BIOS während der Startprozedur an diese Stellen geladen. Obwohl die Kontrolldaten zum BIOS gehören, können Programme darin lesen und sogar Werte ändern. Auch wenn Sie nicht vorhaben, die Informationen eines Teils des BIOS zu nutzen, sollten Sie diesen Abschnitt genau lesen, um sich ein Bild vom Funktionsmechanismus Ihres Computers machen zu können.

Um Verwirrungen zu vermeiden, machen Sie sich bitte noch einmal klar, daß die Speicheradresse hex 400 in segmentierter Schreibweise als 0040:0000 oder wahlweise als 0000:0400 ausgedrückt werden kann. Alle drei Bezeichnungen sind vollkommen gleichwertig und beziehen sich auf dieselbe Speicherstelle.

Der Kontrollinformationsbereich

Einige der Speicherstellen, die in den Gebieten hex 400 und 500 angesiedelt sind, haben für Programmierer eine besondere Bedeutung. Viele enthalten Informationen, die "lebenswichtig" für die BIOS- und DOS-Routinen sind. In den meisten Fällen kann ein Anwendungsprogramm die in diesen Speicherzellen abgelegten Daten durch den Aufruf eines BIOS-Interrupts lesen, in jedem Fall ist ein direkter Zugriff möglich. Die Inhalte dieser Speicherplätze können Sie auf Ihrem eigenen Computer leicht mit dem Befehl DEBUG oder mit einem BASIC-Programm feststellen.
Geben Sie folgende Befehle ein, wenn Sie DEBUG benutzen wollen:

```
DEBUG
D 0:XXXX L 1
```

In dem Beispiel ist xxxx die Hexadresse, die Sie einsehen wollen. L 1 veranlaßt die Anzeige des ersten Byte. Wollen Sie mehrere Bytes ansehen, geben Sie einfach die Anzahl der gewünschten Bytes in hexadezimaler Schreibweise hinter dem L-Befehl ein.
Um die Daten unter BASIC aufzulisten, können Sie das folgende einfache Programm übernehmen. Beachten Sie, daß die Parameter *hexadresse* (hexadezimal) und *byteanzahl* (dezimal) durch aktuelle numerische Werte ersetzt werden müssen.

```
10 DEF SEG = 0
20 FOR 1 = 0 TO byteanzahl - 1
30   VALUE = PEEK(&Hhexadresse + I)
40   IF VALUE < 16 THEN PRINT "0";  für führende Nullen
50   PRINT HEX$ (VALUE);" ";
60 NEXT I
```

Auf den nächsten Seiten finden Sie die wichtigsten Adressen, alle in Hexschreibweise dargestellt.
410 (2-byte-Wort) - Dieses Wort enthält einen Datenwert, der die Ausstattungsliste enthält, die von Routine 17 (hex 11) aufgerufen wird. Das Format dieses Wortes wurde für den Standard-PC und den XT entworfen und kann bei anderen Modellen abgewandelt erscheinen.

Bit																Bedeutung
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
X	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	Anzahl der angeschlossenen Drucker
.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	1, wenn serieller Drucker installiert
.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	1, wenn Spieleadapter installiert
.	.	.	.	X	X	X	.	.	.	.	.	.	.	.	.	Anzahl der seriellen RS-232-Ports
.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	0, wenn DMA-Chip vorhanden; Standard für alle Modelle
.	.	.	.	.	.	.	.	X	X	.	.	.	.	.	.	+ 1 = Anzahl der Laufwerke; 00 = 1 LW; 01 = 2 LW; 10 = 3 LW; 11 = 4 LW; (Siehe Bit 0)
.	.	.	.	.	.	.	.	.	.	X	X	.	.	.	.	Bestimmt den Bildschirmmodus: 10 = 80 Zeichen Farbe, 11 = 80 Zeichen S/W
.	.	.	.	.	.	.	.	.	.	.	.	X	X	.	.	RAM-Kapazität der Hauptplatine: 00 = 16 Kbyte 01 = 32 Kbyte 11 = 64 Kbyte (Standard)
.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	Unbenutzt, sollte 0 sein
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	1, wenn Laufwerk vorhanden, dann siehe Bits 6 und 7

Tabelle 3-2 Die Kodierung der Ausstattungsliste in hex 410

413 (2-byte-Wort) - Dieses Wort enthält die nutzbare Speicherkapazität in Kbyte. Die BIOS-Routine 18 (hex 12) zeigt Ihnen den Wert in diesem Wort an.

417 (2-byte-Wort der Tastaturstatus-Bits) - Mit diesen Bytes wird die Interpretation von Tastaturanschlägen durch die ROM-BIOS-Routinen kontrolliert. Diese Bytes zu verändern, bedeutet, vorgegebenen Funktionen von Tasten eine andere Funktion zuzuweisen. Das Byte in Adresse 417 kann manipuliert werden. Das zweite Byte (Adresse 418) sollten Sie aber lieber nicht verändern. In den Kapiteln 6 und 12 finden Sie weitere Details über die beiden Bytes.

419 (ein Byte) - Dieses Byte ist für einen in Zukunft geplanten alternativen Gebrauch der Tastatur bestimmt.

41A (2-byte-Wort) - Dieses Wort zeigt auf den Anfang des Tastaturpuffers, der bei 41E beginnt. Dort werden Tastenanschläge bis zur Verarbeitung zwischengespeichert.

41C (2-byte-Wort) - Das Wort zeigt auf das Ende des Tastaturpuffers.

41E (32 Bytes, als 16 2-byte-Einheiten genutzt) - Der BIOS-Tastaturpuffer kann bis zu 16 Tastenanschläge zwischenspeichern, bis sie mit Hilfe des BIOS-Interrupts 22 (hex 16) gelesen werden. Der Puffer arbeitet sequentiell, so daß zwei Zeiger für Anfang und Ende (41A und 41C) nötig sind. Die Daten sollten nicht geändert werden.

43E (ein Byte) - Das Byte zeigt an, ob ein Laufwerk rekalibriert werden muß, bevor eine Spur gesucht werden kann. Die Bits 0 bis 3 entsprechen den Laufwerken 0 bis 3. Ist der Wert eines Bit gleich 0, ist eine Rekalibrierung nötig. Sie finden meist den Wert 0 gespeichert, wenn Sie mit einem soeben benutzten Laufwerk Probleme hatten. Ein Bit wird z.B. auf 0 gesetzt, wenn Sie von einem leeren Laufwerk ein Diskettenverzeichnis (DIR) verlangen und dann mit A auf diese Meldung reagieren (je nach der jeweiligen Version von DOS erscheint der Text in englischer oder in deutscher Sprache):

```
Not ready reading error B:     Nicht bereit-Fehler auf Laufwerk B:
Abort, Retry, Ignore?          Abbrechen, Wiederholen, Ignorieren?
```

43F (ein Byte) - Auskunft über den Status des Diskettenlaufwerkmotors. Die Bits 0 bis 3 sind entsprechend den Laufwerken 0 bis 3 zugeordnet. Ist ein Bit 1, so läuft der Motor.
440 (ein Byte) - Dieses Byte enthält den "Count-Down", bis der Diskettenmotor abgestellt wird. Der Wert wird am Anfang jeder Diskettenoperation auf 37 (ca. 2 Sekunden) gesetzt. Bei jedem Taktsignal wird der Wert um 1 heruntergesetzt. Der Diskettenmotor ist abgestellt, wenn der Wert 0 ist.
441 (ein Byte) - Das Byte gibt den Diskettenstatus wieder. Jedes Bit bezieht sich auf einen möglichen Fehler. Ist das Bit gesetzt (1), so ist der Fehler aufgetreten.

Bit 7	6	5	4	3	2	1	0	Bedeutung
X	.	.	.	.	.	.	.	Disketten-Timed-Out: Keine Antwort innerhalb der erwarteten Zeit
.	X	.	.	.	.	.	.	Suche nach Spur erfolglos
.	.	X	.	.	.	.	.	Disketten-Kontroll-Chip ausgefallen
.	.	.	X	.	.	.	.	Prüfsummenfehler (CRC)
.	.	.	.	X	.	.	.	DMA-Diskettenfehler
.	.	.	.	.	X	.	.	Sektor nicht gefunden, Diskette beschädigt oder nicht formatiert
.	.	.	.	.	.	X	.	Adreßmarke auf Diskette nicht gefunden
.	.	.	.	.	.	.	X	Ungültiges Laufwerkskommando

Tabelle 3-3 Die Kodierung des Disketten-Status-Byte in hex 441

442 (sieben Bytes) - Bytes mit Statusinformationen über den Bildschirm-Controller.
Bei hex 449 beginnt ein 30-bit-Bereich für die Bildschirmsteuerung. Das BIOS benutzt diesen Bereich, um wichtige Bildschirminformationen abzulegen. Diese Daten können von Programmen gefahrlos gelesen werden, es ist aber meist risikoreich, sie zu verändern. Eine Änderung kann zu unvorhersehbaren Ergebnissen führen, der Autor hat bereits selbst schon

einige bizarre Resultate erlebt. Die einzigen Bytes in diesem Bereich, die sich ohne Komplikationen manipulieren lassen, sind scheinbar die Cursorpositions-Bytes. Mehr über den Cursor finden Sie in Kapitel 4.8 und unter der Hexadresse 450.

449 (ein Byte) - Ein Wert von 0 bis 7, 10 oder von 13 bis 15 spezifiziert den augenblicklich gewählten Bildschirmmodus. Es handelt sich dabei um dieselbe Bildschirmmodus-Kodierung, die von den BIOS-Routinen für die Bildschirmbefehle benutzt wird. Mehr über diese Unterprogramme und allgemeine Informationen über die Bildschirmmodi finden Sie in den Kapiteln 9 und 4.2.

Folgende Programmzeilen lesen dieses Byte, um den Bildschirmmodus festzustellen:

```
DEF SEG = 0              'Register DS auf 0 setzen
BILDMODUS = PEEK(&H449)  'Adresse hex 449 auslesen
```

In Kapitel 4.3.1 finden Sie einige Besonderheiten der Modi 4 und 5 in BASIC .

Code	Bedeutung	Code	Bedeutung
0	40-Spalten-Text, keine Farbe	13	mittelauflösende Grafik, 16-farbig (nicht mit Standard-Farb-/Grafikadapter)
1	40-Spalten-Text, 16-farbig	14	hochauflösende Grafik, 16-farbig (nicht mit Standard-Farb-/Grafikadapter)
2	80-Spalten-Text, keine Farbe	15	spezielle hochauflösende Grafik, 4-farbig (nicht mit Standard-Farb-/Grafikadapter)
3	80-Spalten-Text, 16-farbig		
4	mittelauflösende Grafik, 4-farbig		
5	mittelauflösende Grafik, keine Farbe (4 Grautöne)		
6	hochauflösende Grafik, 2-farbig		
7	Monochromadaptermodus		

Tabelle 3-4 Die Kodierung des Bildschirmmodus-Bytes in hex 449

44A (2-byte-Wort) - Angabe zur Bildschirmbreite in Textspalten. Die Spaltenbreite wird im Hexäquivalent von 20, 40 oder 80 Spalten gespeichert. Der Bildschirmmodus 8, der Modus für niedrige Auflösung, hat eine Textbreite von 20.

44C (2-byte-Wort) - Hier steht die Bildschirmlänge, das heißt, die Anzahl der Bytes, die für eine Bildschirmseite benötigt werden. Dies variiert mit den verschiedenen Modi.

44E (2-byte-Wort) - Der Bildschirmpositions-Offset ist die Offsetadresse der momentan angezeigten Seite im Bildschirm.

450 (acht 2-byte-Worte) - Diese Worte geben die Cursorposition für acht veschiedene Bildschirmseiten an, beginnend mit Seite 0. Das erste Byte eines jeden Wortes spezifiziert die Spalte (0 bis 19, 39 oder 79), das

zweite Byte die Zeile (0 bis 24). Die Cursorposition kann durch Verändern dieser Information gesteuert werden. Für Programmiersprachen, die keine eingebaute Cursorkontrolle besitzen, ist dies eine elegante Möglichkeit, auf den Cursor Einfluß zu nehmen, ohne eine Routine auf Assemblerebene zu bemühen, die den BIOS-Routinen hinzugefügt werden müßte.
Beim Verändern der Daten erfolgt der Wechsel des Cursors nicht sofort, sondern erst bei der nächsten Bildschirmausgabe. Starten Sie einmal DEBUG und geben dann folgendes Kommando ein:

```
F 0:450 L 2 8 8
```

Der Cursor springt in Zeile 8, Spalte 8, nachdem die Return-Taste gedrückt wurde. Daß dies keine gute Programmiertechnik ist, braucht nicht erwähnt zu werden. Dennoch ist es nützlich, über die Möglichkeit informiert zu sein.
460 (2-byte-Wort) - Diese beiden Bytes beinhalten den Wert der Cursorgröße in Rasterzeilen. Das erste Byte gibt die Endrasterzeile, das zweite die Anfangsrasterzeile an. Im Gegensatz zum Cursorpositionsfeld bewirkt eine Veränderung dieser Werte nicht automatisch auch eine Änderung des Cursors.
462 (ein Byte) - Dieses Byte enthält die Nummer der momentan angezeigten Bildschirmseite.
463 (2-byte-Wort) - Adressenangabe des 6845-Bildschirm-Controllerbausteins. Normalerweise steht hier der Wert hex 3D4.
465 (ein Byte) - In diesem Byte wird der Wert des Bildschirmmodus abgelegt.
466 (ein Byte) - Hier findet sich die Bitmaske für die Farbpaletten.
Mehr über Farbpaletten steht in Kapitel 4.3.
467 (fünf Bytes) - Diese Bytes werden zur Steuerung eines Kassettenrecorders benutzt.
46C (vier Bytes abgelegt als 2-byte-Worte, die aber wie ein 4-byte-Wort behandelt werden) - Dieser Speicherbereich wird als "Hauptzähler" des Computers verwendet; bei jedem Systemtakt wird der Zählerstand um eins erhöht. Ausgangspunkt ist dabei die Annahme, daß die Zählung um Mitternacht, also null Uhr, begann. Erreicht nun der Zählerstand den 24-Stunden-Wert, wird die Speicherstelle hex 470 gesetzt und die Zählung beginnt wieder bei Null. DOS und BIOS berechnen die aktuelle Zeit entweder aus diesem Zählerstand oder setzen den Wert durch entsprechende Belegung des Feldes. Der Wert wird durch Interrupt 26 (hex 1A) gemeldet und gesetzt.
470 (ein Byte) - Dieses Byte zeigt an, das der maximale Zählerstand (entspricht 24 Stunden) überschritten und folglich wieder neu angefangen wurde, von Null ab zu zählen. In diesem Fall sitzt Byte 470 auf eins und signalisiert damit, daß das Datum erhöht werden muß (24 Stunden entsprechen schließlich einem Tag). Der Wert wird durch die Taktimpulsrou-

tine gesetzt und zeigt an, daß Mitternacht vorüber ist. Das Byte wird automatisch wieder auf Null gesetzt, wenn es durch Interrupt 26 (hex 1A) gelesen wird, da jedes Programm, das dieses Feld liest, auch das Datum aufdatieren muß.

Hinweis: **Dieses Byte wird an Mitternacht automatisch auf 1 gesetzt und danach nicht mehr weitergezählt. Es gibt somit keine Möglichkeit festzustellen, ob das "Ereignis Mitternacht" schon mehrmals stattgefunden hat, solange die Uhrzeit nicht ausgelesen wurde.**

471 (ein Byte) - Hiermit wird eine Tastaturunterbrechung unter BIOS angezeigt. Steht das Bit 7 auf 1, wurde die Unterbrechungstastenkombination (Ctrl-C oder Ctrl-Break) gedrückt.
472 (2-byte-Wort) - Dieses Byte signalisiert, daß ein Tastaturneustart vorliegt. Wenn das System durch die Tastenkombination Ctrl-Alt-Del neu gestartet (man sagt auch *gebootet*) wird, nimmt dieses Wort den Wert hex 1234 an, solange der Startvorgang andauert. Dies darf man wohl eher als Kuriosität verstehen, denn wozu soll eine solche Information wohl nützlich sein?
473 (ein Byte) - Für IBM reserviert. Dieses Byte wird auf hex 24 gesetzt, wenn IBM ihren Aktionären Dividende zahlt.
500 (ein Byte) - In diesem Byte kann der Status eines Bildschirmdruckvorganges (*Hardcopy*) erkannt werden. Drei Werte sind möglich (hexadezimal):
00 OK-Status
01 Bildschirmdruckoperation im Ablauf
FF Fehler während der Ausgabe des Bildschirms auf einen Drucker
504 (ein Byte) - DOS benutzt dieses Byte, wenn mit nur einem Diskettenlaufwerk zwei Laufwerke vorgetäuscht werden sollen, wie etwa beim XT. Der Wert zeigt an, ob das reale Laufwerk als Laufwerk A oder B fungiert.
00 Aktiviert als Laufwerk A
01 Aktiviert als Laufwerk B
510 (ein 2-byte-Wort) - Dieser Bereich wird vom BASIC benutzt und enthält den Standard-Datensegmentwert (DS).
BASIC erlaubt uns die Festlegung eines eigenen Datensegmentwertes mit Hilfe der Anweisung DEF SEG = Wert. Der Offset in diesem Segment wird durch PEEK- oder POKE-Funktionen gesetzt. Wir können das Datensegment wieder zurücksetzen, indem wir die Anweisung DEF SEG ohne Wert verwenden. Da BASIC keinen Befehl zur Verfügung stellt, der den Wert dieser Speicherstelle anzeigt, können wir dieses kleine Unterprogramm benutzen:

```
DEF SEG = 0
DATENSEGMENT = PEEK(&H511) * 256 + PEEK(&H510)
```

Hinweis: **BASIC verwaltet seine eigenen internen Daten auf der Basis des Datensegmentwertes. Jede Änderung kommt daher einer "Sabotage" der gesamten BASIC-Operationen gleich.**

512 (vier Bytes) - Dieser Bereich wird von BASIC als Interruptvektor benutzt, der auf die Uhrroutine zeigt.

Hinweis: Um seine Aufgabe schneller erledigen zu können, läuft BASIC mit der vierfachen Systemgeschwindigkeit. Daher muß die BIOS-Systemtaktroutine durch eine andere ersetzt werden. Die BIOS-Routine wird von BASIC mit der normalen Frequenz aufgerufen, also alle viermal pro BASIC-Takt. Lesen Sie auch in Kapitel 8 nach.

516 (vier Bytes) - Dieser Bereich wird von BASIC als Interruptvektor benutzt und zeigt auf die Tastaturunterbrechungsroutine des BASIC.
51A (vier Bytes) - Dieser Bereich wird von BASIC ebenfalls als Interruptvektor benutzt. Er zeigt auf die Diskettenfehlerbehandlungsroutine des BASIC.

3.2.2.1 Speicherbereiche für interne Kommunikation

Die BIOS-Kontrollinformationen umfassen den größten und auch den wichtigsten Teil des Speicherbereiches im Adreßblock 400. Dennoch muß noch eine andere Nutzung des Blocks erwähnt werden. In den Speicherzellen hex 47F bis hex 4FF liegt die sog. *Intra-Applications Communications Area* (ICA), das ist ein 16-byte-Bereich, der zur internen Kommunikation vorgesehen ist. Man spricht daher auch vom *internen Kommunikationsbereich.* Der ICA wird zur Ablage solcher Daten verwendet, die von mehreren Programmteilen benutzt werden. Das ist sehr nützlich für Programme, die eigenständig als DOS-Programmbestandteile ablaufen und Daten für andere Programme, die zur Erfüllung einer Gesamtaufgabe zusammen benutzt werden, hinterlassen müssen. Der Gebrauch dieses Speicherbereichs ist allerdings sehr selten anzutreffen.
Da eine beliebige Anzahl von Programmen Daten im ICA ablegen darf, können leicht wichtige Informationen überschrieben werden. Wenn Sie den ICA benutzen wollen, rate ich Ihnen, eine Prüfsumme einzuführen und eine Markierung der Daten, so daß Sie sie immer als die Ihren erkennen können und sicher sind, daß sie nicht geändert wurden.

Vorsicht: Der ICA ist definitiv in den 16 Bytes von hex 4F0 bis 4FF untergebracht. Die Angabe der Position hex 500 bis 5FF in einigen Publikationen ist falsch.

3.2.3 ROM-Versionen und Modellidentifikation

Da das BIOS im Nur-Lese-Speicher abgelegt ist, sind Änderungen des BIOS für Erweiterungen oder Korrekturen nicht einfach durchführbar. Die ROM-Programme müssen also sehr gut ausgetestet werden, bevor sie in den Speicherchips "festgefroren" werden. Dennoch besteht natürlich immer die Möglichkeit, daß sich auch schwerwiegende Systemfehler in

ein ROM einschleichen. Bei den IBM PCs waren bislang glücklicherweise stets nur kleine Mängel festzustellen, die in den jeweils neueren Geräten behoben wurden. Dadurch entstanden im Laufe der Zeit aber verschiedene ROM-Versionen, die sich in kleinen Details unterscheiden.
Es gibt noch einen anderen Grund, warum heutzutage unterschiedliche ROM-Versionen verbreitet sind: Schließlich muß das ROM an die jeweiligen Geräte angepaßt werden (z.B. an den AT gegenüber dem Standard-PC).
Die unterschiedlichen Versionen der IBM-ROM-Software stellen für Programmierer, die bemerken, daß ihre Programme auf anderen Geräten anders verarbeitet werden, eine Herausforderung dar. Eine wesentlich bedeutendere Herausforderung liegt allerdings in der Feststellung, das bei manchen PC-Modellen ein anderer Satz von ROM-BIOS-Routinen als der der Standard-Ausführung des PC zur Verfügung steht. Dies ist z.B. beim AT der Fall, wie bereits angesprochen.
Die Sicherheit, daß ein selbständig entwickeltes Programm auf dem ausgewählten Computer und mit den entsprechenden ROM-Programmen arbeitet, gewähren uns zwei Identifikationsmerkmale, die am Ende jedes ROM-Speichers untergebracht sind. Zum einen findet sich dort das Freigabedatum des BIOS, d.h., das Datum, ab dem diese BIOS-Version zur Auslieferung kam. Dies kann natürlich zur Identifikation des benutzten ROM-Satzes verwendet werden kann. Zum anderen existiert ein Idenfikationsmerkmal, das das jeweilige Modell kennzeichnet. In Original-IBM-PCs sind diese beiden Bezeichnungen (Freigabedatum und Modellidentifikation) immer vorhanden, für die zahlreichen Kompatiblen gilt dies nur in eingeschränktem Maße.
Das ROM-Freigabedatum finden Sie in dem 8-byte-Speicherbereich von F000:FFF5 bis F000:FFFC (zwei Bytes vor dem Modellidentifikations-Byte). Es besteht aus ASCII-Zeichen im üblichen amerikanischen Datumsformat, z.B. 06/01/83 steht für 01. Juni 1983.
Der einzige Nutzen, der in diesem Freigabedatum liegt, ist die Überprüfbarkeit der einzelnen ROM-Versionen. Falls ein Programm auf zwei scheinbar gleichen Geräten unterschiedlich abläuft, sollten Sie unbedingt das ROM-Datum untersuchen. Die meisten Programme nehmen darauf allerdings keine Rücksicht, sondern orientieren sich ausschließlich am Modellidentifikations-Byte (Modell-ID-Byte), um die unterschiedlichen PC-Varianten zu erkennen.
Mit den Befehl DEBUG können Sie das Freigabedatum auslesen:

```
DEBUG
D F000:FFF5 L 8
```

In einem BASIC-Programm kann das Auslesen wie folgt geschehen:

```
10 DEF SEG = &HF000
20 FOR I = 0 TO 7
30   PRINT CHR$(PEEK(&HFFF5+I));
40 NEXT
50 END
```

Hier eine Tatsache, auf die Sie vielleicht auch stoßen werden: P. Norton betreibt drei PCs und jeder davon besitzt ein anderes ROM. Einer hat die 04/24/81-Version, ein anderer hat die Version vom 10/19/81 und der dritte schließlich verfügt über die Version 10/27/82.

Freigabedatum	Modell
04/24/81	Original PC
10/19/81	geänderter PC (einige Fehler wurden beseitigt)
08/16/82	Original XT
10/27/82	Aufrüstung des PC zur XT-BIOS-Ebene
11/08/82	Original Portable-PC
01/10/84	Original AT

Tabelle 3-5 Die Freigabedaten der unterschiedlichen ROM-Versionen

Die Modellidentifikation (Modell-ID) ist ein Byte in Speicherstelle F000:FFFE. Die nachfolgende Tabelle zeigt eine Aufstellung der allgemein bekannten ID-Werte für drei IBM-PC-Modelle. Es ist zu erwarten, daß diesem Muster bei der Weiterentwicklung bzw. der Herausgabe neuer Rechner gefolgt wird.

ID-Wert		Modell
Dez	Hex	
255	FF	PC (Original IBM Personal Computer)
254	FE	XT und Portable-PC
252	FC	AT

Tabelle 3-6 Die Modellidentifikationen für drei verschiedene IBM PC-Modelle

Sie werden bemerken, daß die Adresse 253 (hex FD) fehlt; das ist die ID-Nummer des in Deutschland unbekannten PCjr, der aus diesem Grunde im vorliegenden Buch auch nicht behandelt wird.
Die angegebene Codierung ist nicht eindeutig. Der PC XT, der ursprünglich den ID-Wert hex FE hatte, wird mittlerweile oft mit dem Code des PC (hex FF) angetroffen. Andererseits ist der Code des XT (hex FE) ebenfalls für den tragbaren PC (Portable-PC) gültig. Derartige Bezeichnungen der Firma IBM sind also nicht fest zementiert. Dennoch kann man

davon ausgehen, daß es eine einfache Regel gibt, um die Modellbezeichnungen richtig deuten zu können. Sind die Unterschiede zweier Modelle so groß, daß ein Programm in der Lage sein muß, eindeutig das Modell zu erkennen, dann sind die angegebenen ID-Kennzeichen zuverlässig zu beachten. Der AT ist ein solches Gerät. Sind die Unterschiede zweier Modelle aber nur geringfügig, wie zwischen dem Original-PC und dem Standard-PC, dem manchmal auch *PC-2* genannten Nachfolger (der 256 Kbyte Speicher auf seiner Systemplatine akzeptiert), dem XT und dem tragbaren PC, so können die Bezeichnungen variieren. Für alle praktischen Zwecke kann man kurzerhand sagen: Die Kennzeichen FE und FF identifizieren, mehr oder weniger, den Standard-PC.
Es ist grundsätzlich möglich, daß kompatible Rechner mit derselben Methode identifiziert werden können. Darüber gibt es aber keine zuverlässigen Informationen.
Mit Hilfe von DEBUG können Sie folgendermaßen die Modellkennzeichnungen einsehen:

```
DEBUG
D F000:FFFE L 1  ; zeigt das eine Byte der spezifizierten Stelle
```

Das folgende Programm liest das Byte zum Identifizieren:

```
10 DEF SEG = &HF000          'definiert Segment F000 im DX-Register
20   IF PEEK(&HFFFE) = 254 THEN PRINT "Ich bin ein  XT"
30   IF PEEK(&HFFFE) = 255 THEN PRINT "Ich bin ein Standard-PC"
40   IF PEEK(&HFFFE) = 252 THEN PRINT "Ich bin ein AT"
50 END
```

3.3 ROM-BASIC

Wir kommen nun zum dritten Element des ROM, dem ROM-BASIC. Das ROM-BASIC hat zwei Aufgaben. Erstens stellt es das Grundgerüst der Programmiersprache BASIC zur Verfügung, das die meisten Befehle und das Fundament, z.B. die Speicherverwaltung, die BASIC benutzt, umfaßt. Die Diskettenversionen, die wir in BASIC.COM und BASICA.COM finden, sind Erweiterungen des ROM-BASIC und in vielen Funktionen auf diesen Nukleus angewiesen. Als zweites stellt das ROM-BASIC das *Kassetten-BASIC* von IBM bereit. Das ist das BASIC, das benutzt wird, wenn dem Computer kein Disketten-BASIC zur Verfügung steht. In der Praxis ist dieses BASIC allerdings bedeutungslos, da es fast niemand verwendet.
Wann immer wir eines der interpretierenden BASICs benutzen, das Kassetten-BASIC oder eines der Disketten-BASICs (BASIC oder BASICA), das ROM-BASIC wird ebenfalls benutzt, wenn wir davon auch nichts merken. Kompiliertes BASIC macht hingegen vom ROM-BASIC keinen Gebrauch.

3.4 ROM-Erweiterungen

Das vierte Element des ROM hat mehr mit der Entwurfsphilosophie, die dem PC zugrunde gelegt wurde, zu tun, als mit den tatsächlichen Speicherinhalten. Der PC bietet zwei Erweiterungsmöglichkeiten der eingebauten ROM-Software an: einmal permanente (feste) Erweiterung des ROM-BIOS und zum anderen den Anschluß sog. *ROM-Kassetten*, auch *Kartuschen* oder *Cartridges* genannt. Für beides (feste Erweiterungen und ROM-Kassetten) ist ein festgelegter Speicherbereich reserviert.
Permanente ROM-BIOS-Erweiterungen sind Programme, die wie das eingebaute ROM-BIOS arbeiten, aber darüber hinausgehende Möglichkeiten zum Einsatz anbieten. Im allgemeinen sind dies Unterstützungsprogramme für neue Peripheriegeräte. Das beste Beispiel dieser ROM-Erweiterungen ist das Hilfsprogramm für das mit dem XT eingeführte Festplattenlaufwerk, aber auch der HR-Farb-/Grafikadapter (*Enhanced Graphics Adapter*) kann hier angeführt werden. Da das Original-ROM-BIOS unmöglich die Unterstützungsroutinen für zukünftige Geräteentwicklungen bereitstellen konnte, sind diese ROM-Erweiterungen sehr nützlich.
Zwei Speicherbereiche werden für diese permanenten ROM-Erweiterungen verwendet. Der eine ist der unbenutzte Rest des F-Blockes, dessen Größe leider von Modell zu Modell variiert. Bei den meisten Modellen ist das Gebiet mit den Segmentadressen von F000 bis F600, das 24 Kbyte umfaßt, verfügbar. Der andere Speicherbereich geht von C000 bis zu CFFF, umfaßt also den gesamten C-Block. Die ROM-BIOS-Routine, die für die Festplatte des XT zuständig ist, beginnt bei der Segmentadresse C800, der HR-Farb-/Grafikadapter (*Enhanced Graphics Adapter*) von IBM bei C000. Obwohl die Speicherbereiche der permanenten ROM-Erweiterungen von IBM klar festgelegt sind, ist doch die Gefahr eines Konfliktes mit Routinen anderer Hersteller immer gegeben.
Normalerweise sind diese permanenten Speichererweiterungen "halbfest" am Computer angeschlossen bzw. eingebaut, entweder in einer Erweiterungskarte oder in einem ROM-Chip, der in einen entsprechenden Sockel direkt auf der Hauptplatine des Computers eingesteckt wird.
Kassetten-ROMs, die einfach und schnell angeschlossen werden können, arbeiten ähnlich wie Disketten: Sie laden zeitweilig Programme für spezielle Zwecke in den Hauptspeicher. Zur Klarstellung: Eine ROM-Kassette hat überhaupt nichts mit herkömmlichen Kassetten (Magnetspeicherbändern) zu tun, sondern ist ein Modul aus Platine mit Gehäuse, das ROM-Chips beinhaltet und an den Computer angesteckt werden kann, sofern dieser einen entsprechenden Anschluß aufweist. Der für ROM-Kassetten zur Verfügung stehende Speicherplatz beträgt 128 Kbyte und umfaßt die Blöcke D und E. Diese Form der ROM-Erweiterung ist in der Praxis ausschließlich beim PCjr anzutreffen. Erstens bringt nur der PCjr einen speziell dafür ausgelegten Einschubschacht für ROM-Kassetten mit, zum anderen gibt es eigentlich keinen vernünftigen Grund, mit ROM-

Kassetten zu arbeiten, wenn man eine Diskettenstation zur Verfügung hat. Grundsätzlich ist aber jeder IBM PC von der technischen Konzeption her in der Lage, ROM-Kassetten zu unterstützen.
Beide Arten der Erweiterung werden beim Startprozeß mit berücksichtigt. Um ROM-Erweiterungen zu finden, beginnt das Standard-ROM bei C000 und untersucht alle 2 Kbyte-Blöcke auf das Kennzeichen für ROM-Erweiterungen (hex 55 AA). Wird ein solches Kennzeichen gefunden, so übergibt das Startprogramm die Kontrolle an die ROM-Erweiterung, damit diese alle zu ihrer Integration nötigen Maßnahmen veranlassen kann. An dieser Stelle kann z.B. die Kontrolle über den gesamten Prozessor von der Routine übernommen werden. Einige Routinen arbeiten mit dieser Methode, die meisten erledigen aber trivialere Dinge, wie das Überprüfen der ihnen zugeordneten Geräte. Ist die Initialisierung abgeschlossen, übergibt die Erweiterung die Kontrolle an das Startprogramm, das in seiner Aufgabe an dieser Stelle fortfährt.

3.5 Anmerkungen

Eine Auflistung der ROM-Software könnte Bände füllen und tatsächlich nimmt sie auch einen guten Teil des Umfanges der Technischen Referenzhandbücher von IBM ein. Dennoch sieht IBM die direkte Nutzung der dort gegebenen Informationen nicht gerne. Es kann sehr lustig werden, wenn man die ROM-Programme auf gut Glück hin ausprobiert. In diesem Buch wird immer ausgeführt, ob die Verwendung gefährlich werden kann oder ob zumindest Sicherheitsvorkehrungen getroffen werden müssen. Sie sollten diese Ausführungen jeweils mit besonderer Aufmerksamkeit lesen.

Kapitel 4
Der Bildschirm

4.1 Bildschirmadapter 66
4.1.1 Speicher und Bildschirmadapter 67
4.1.2 Anlegen des Bildschirmabbildes 68
4.2 Bildschirmmodi 69
4.2.1 Bildschirmauflösung 70
4.2.2 Steuerung des Bildschirmmodus 71
4.3 Farben 72
4.3.1 Farbunterdrückende Modi 74
4.3.2 Farbe in Text- und Grafikmodus 75
4.3.2.1 Setzen der Farben in die Textmodi 75
4.3.2.2 Setzen der Attribute im Monochrommodus 77
4.3.2.3 Setzen der Farben in die Grafikmodi 77
4.3.2.4 Veränderbare Farbpaletten des HR-Farbgrafikadapters (EGA) 79
4.4 Aufbau des Bildschirmspeichers 79
4.4.1 Handhabung mehrerer Seiten in den Textmodi 80
4.4.2 Seitendarstellung in den Grafikmodi 81
4.4.3 Zeichendarstellung in den Text- und Grafikmodi 82
4.4.3.1 Speichern von Zeichen in den Textmodi 82
4.4.3.2 Speichern von Pixel in den Grafikmodi 83
4.5 Bildschirmsteuerung 84
4.5.1 Direkte Hardwarekontrolle 86
4.6 Kompatibilitätserwägungen 89
4.7 Allgemeines über Monitore 90
4.8 Der Cursor 92

Für viele Menschen ist der Bildschirm identisch mit dem Computer. Software wird allzu oft nach dem optischen Eindruck des Programmablaufs beurteilt. In diesem Kapitel werden wir uns mit den verschiedenen Bildschirmmodi beschäftigen und damit, wie sie entstehen. Dabei werden Sie lernen, wie der Bildschirm zu manipulieren ist, um alle möglichen Effekte zu erzeugen.

4.1 Bildschirmadapter

Um eine Bildschirmanzeige zu erzeugen, brauchen die meisten PC-Modelle (einschließlich des PC, XT und AT) einen Bildschirmadapter - eine spezielle Platine, die normalerweise an einen Erweiterungssteckplatz des Computers angeschlossen wird. Modelle, wie den Portable-PC oder den Compaq-Rechner erhält man bereits mit Bildschirmadaptern ausgestattet, die aber austauschbar sind.
Der Bildschirmadapter verbindet Computer und Bildschirmmonitor mit einem *CRT-Controller-Chip* (*Cathode Ray Tube*, Kathodenstrahlröhre), zu deutsch meistens *Bildschirm-Controller* genannt. Der Adapter beinhaltet auch einen Satz programmierbarer Ein-/Ausgabe-Ports, einen ROM-Zeichengenerator sowie RAM-Speicher, um die Bildschirminformationen ablegen zu können.
Die Vielzahl von Adaptern, die auf dem Markt angeboten werden, sind alle den zwei Originalen von IBM nachgebildet, dem Farb-/Grafik und dem Monochromadapter. Der Schwerpunkt dieses Kapitels liegt auf diesen zwei Adaptern, Anmerkungen zu anderen finden Sie aber auch im Text.
Bildschirmanzeigen werden für zwei unterschiedliche Modi produziert, nämlich den Textmodus und den Grafikmodus, wie sie von IBM bezeichnet werden. Der Textmodus stellt nur komplette Zeichen dar, obwohl er mit Hilfe seines Zeichenvorrates einfache Grafiken durchaus anfertigen kann. Lesen Sie hierzu bitte auch Anhang C, dort finden Sie mehr über Zeichen. Der Grafikmodus ist in erster Linie für das Darstellen komplexer Grafiken entwickelt worden, er kann aber auch eine Reihe von Textzeichen unterschiedlicher Form und Größe darstellen.
Der Farb-/Grafikadapter kann in beiden Modi arbeiten, um Texte und Grafiken in unterschiedlichen Formaten und Farben zu produzieren. Er wurde entworfen, um mit jedem beliebigen Bildschirm zu arbeiten, vom einfachen Fernsehbildschirm bis zum hochauflösenden Farbmonitor.
Der Monochromadapter kann im Gegensatz dazu nur im Textmodus arbeiten, er verwendet einen fest gespeicherten Satz von ASCII-alphanumerischen- und Grafikzeichen und erlaubt nur einfarbige Darstellungen. Er ist vor allem für den Geschäftsbereich entwickelt worden und kann nur mit dem IBM-Monochrommonitor (oder einem Äquivalent), der ein spezieller hochauflösender Bildschirmmonitor ist, arbeiten. Mehr über Monitore finden Sie in Kapitel 4.7. Professionelle Benutzer und die Ge-

schäftswelt ziehen den Monochrommonitor vor, weil auf ihm Texte leichter ablesbar sind. Auf der anderen Seite wird heute oft behauptet, daß Farbe und Grafik unerläßlich seien.
Um die genannten Grenzen zu überschreiten, haben verschiedene Hersteller Variationen des IBM-Monochromadapters entwickelt, wie z.B. den Hercules-Bildschirmadapter. Dieser kombiniert die Grafikfähigkeiten (nicht die Farbe) des Farb-/Grafikadapters mit der in der Qualität höher liegenden Textanzeige des Monochromadapters und fügt noch einige Besonderheiten hinzu. Die resultierende Qualität der Grafik ist sogar besser als die des Farb-/Grafikadapters. Der HR-Farbgrafikadapter (HR steht für *High Resolution*, hohe Auflösung) von IBM, meist nach der englischen Bezeichnung *Enhanced Graphics Adapter* kurz EGA genannt, erzeugt auf eine gleichartige Weise Grafiken auf dem Monochrombildschirm.
Ungefähr zwei Drittel aller PCs sind mit dem Standard-Monochromadapter ausgerüstet und können daher keine Grafiken oder Farben erzeugen. Obwohl in der Benutzung von Farben und Grafiken Vorteile liegen, kommen die meisten PCs auch ohne diese Möglichkeit gut zurecht. Sollten Sie Software entwickeln, so beachten Sie also, daß die meisten Computer nur Text anzeigen.
Der IBM-Monochromadapter verfügt nur über einen Teil der Eigenschaften, die der Farb-/Grafikadapter bereitstellt. Aus diesem Grunde werden wir den Farb-/Grafikadapter ausführlich behandeln, der Monochromadapter weicht nur in einigen Details ab. Außerdem werden Sie die Unterschiede zum HR-Farbgrafikadapter (EGA) kennenlernen.

4.1.1 Speicher und Bildschirmadapter

Der Bildschirmspeicher ist physikalisch mit den anderen Anzeigebauelementen auf der Adapterkarte untergebracht. Logisch (vom Standpunkt der CPU aus) ist er aber ein Teil des adressierbaren Hauptspeichers. Die vollen 128 Kbyte Speicherbereich der Blöcke A und B werden für Bildschirmzwecke reserviert, d.h., hex A0000 bis BFFFF kann als Adapterspeicher benutzt werden. Die zwei Original-Bildschirmadapter benötigen nur zwei kleine Bereiche des ihnen zur Verfügung stehenden Speicherplatzes. Der Monochromadapter belegt 4 Kbyte ab der Segmentadresse hex B000, der Original-Farb-/Grafikadapter verfügt über einen Bereich von 16 Kbyte, der bei hex B800 beginnt. Der verbleibende Speicherplatz, insbesondere die 64 Kbyte von hex A000 bis B000, wird für eventuelle Erweiterungen (HR-Farbgrafikadapter) zurückbehalten.
Die spezielle Dual-Port-Architektur des Bildschirmadapterspeichers ermöglicht der CPU und dem Bildschirm-Controller gleichermaßen den Zugang zum Bildschirmspeicher. Tatsächlich kann der Zugriff sogar zum selben Zeitpunkt stattfinden.

4.1.2 Anlegen des Bildschirmabbildes

Sowohl der Monochrom- als auch der Farb-/Grafikadapter speichern Bildschirminformationen *speicherbezogen* (*Memory-Mapped*), d.h., jeder Adresse im Bildschirmspeicher kann eine bestimmte Stelle auf dem Bildschirm zugeordnet werden.
Der Anzeigebaustein liest die Informationen ständig aus dem Speicher aus und gibt sie an den Bildschirm weiter. Die Information kann so schnell geändert werden, wie der Computer neue Informationen von Programmen in den Speicher schreiben kann. Der Bildschirm-Controller ist das Verbindungsstück zwischen Bildschirmspeicher und Monitor. Er übersetzt den Fluß der Bits aus dem Speicher in Lichtpunkte an bestimmten Stellen des Bildschirms.
Diese Lichtpunkte - *Pixel* genannt - werden von einem Elektronenstrahl erzeugt, der die phosphoreszierende Oberfläche des Bildschirms zum Leuchten anregt. Der Elektronenstrahl wird von einer Elektronenkanone (Emitter) generiert und bewegt sich Zeile um Zeile (rasterzeilenweise) über den gesamten Schirm. Dabei hält sich der Strahl waagerecht und senkrecht auf einem genau festgelegten Weg und beschreibt so ein sog. *Raster* auf dem Bildschirm. Der Bildschirm-Controller erzeugt gleichzeitig ein Signal, das den Elektronenstrahl, entsprechend dem Muster der Bits im Speicher, an- und ausschaltet.
Der Bildschirmbaustein frischt den gesamten Bildschirm 60 mal in der Sekunde neu auf, d.h., schreibt die Informationen aus dem Bildschirmspeicher links oben beginnend rasterzeilenweise neu auf die Leuchtschicht

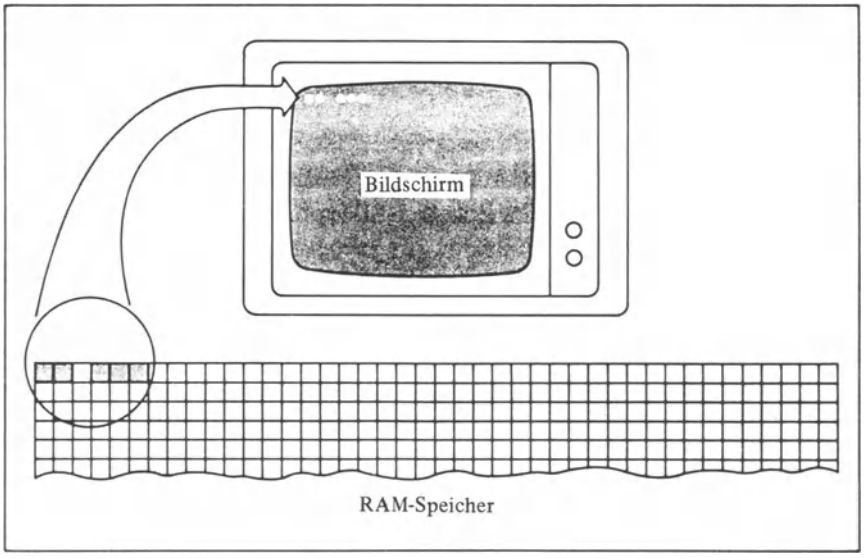


Bild 4-1 Die speicherbezogene Bildschirmdarstellung

des Schirms. Die wechselnden Bilder sehen aufgrund der hohen Wiederholungsgeschwindigkeit klar und stetig aus. Am Ende jedes Auffrischungszyklus (*Refresh Cycle*) muß der Elektronenstrahl von der unteren rechten Ecke zur oberen linken Ecke gelangen, um dort erneut einen Zyklus zu beginnen. Diese Bewegung wird *vertikaler Rücklauf* (*Vertical Retrace*) genannt. Während des Rücklaufes ist der Elektronenstrahl ausgeschaltet und es kann keine Information auf den Schirm geschrieben werden.
Der vertikale Rücklauf dauert genau 1,25 Millisekunden. Diese periodische Leerlaufzeit ist für den Programmierer von großer Bedeutung. Greift die CPU auf den Bildschirmspeicher zu, während der Bildschirm-Controller den Bildschirm beschreibt, so kann ein kurzes Flimmern - auch *Schnee* genannt - auf dem Schirm zu sehen sein. Wenn die CPU aber nur während des vertikalen Rücklaufs in den Speicher schreibt (vom Bildschirm-Controller erfolgt in dieser Zeit kein Zugriff), dann kann dieser Schnee-Effekt vermieden werden. Systeme, die den Farb-/Grafikadapter verwenden, können hierfür ein Status-Bit benutzen, das *Vertical Sync-Signal* genannt wird. Es ist in einem der Ein-/Ausgabe-Ports zu finden (hex 3DA). Dieses Bit wird am Anfang jedes Rücklaufs gesetzt und am Ende wieder gelöscht. Während der 1,25 Millisekunden kann der Prozessor nun so viel Daten wie nur möglich in den Bildschirmspeicher bringen. Am Ende des Rücklaufs kann der Bildschirm-Controller die Daten ohne unerwünschte Nebeneffekte auf den Bildschirm schreiben. Diese Technik ist besonders für das schnelle Verändern von Bildern geeignet.

4.2 Bildschirmmodi

Ursprünglich gab es acht Bildschirmformate, Bildschirmmodi oder Videomodi, die von IBM definiert waren. Weitere sieben und noch mehr wurden zwischenzeitlich hinzugefügt. Der Bildschirmmodus definiert die Eigenschaften der Anzeige einschließlich der maximalen Länge des darstellbaren Textes, der Auflösung oder sonstiger Details im Grafikbereich sowie die Farben. Der Farb-/Grafikadapter vereint in sich mehrere wählbare Formate für Text- und Grafikmodus. Der Monochromadapter bietet nur ein einziges einfarbiges Textformat. Der HR-Farbgrafikadapter (EGA) unterstützt eine Vielzahl alter und neuer Modi.
Jeder Bildschirmmodus trägt eine Nummer von 0 bis 16. Die Modi 0 bis 3 sind die Text- und 4 bis 6 die Grafikmodi des Farb-/Grafikadapters. Modus 7 kann nur vom Monochromadapter verwendet werden, es handelt sich um einen einfarbigen Textmodus. Die Modi 13 bis 16 beziehen sich auf den HR-Farbgrafikadapter (EGA), der auch die Modi 0 bis 7 verwendet.
Farben können in jedem Modus verwendet werden, außer natürlich im Falle der Nummer 7, dem Textmodus des Monochromadapters. Je nach Modus des Farb-/Graphikadapters stehen zwei bis maximal 16 Farbkom-

binationen zur Verfügung, einschließlich einer Palette von Grautönen, die zum *farbunterdrückenden Modus* gehören. Obwohl beim Monochromadapter keine Farbdarstellung möglich ist, existieren doch Abhebungsmöglichkeiten: hohe und normale Intensität, Unterstreichen und umgekehrte (invertierte) Darstellung. Die Verwendung von Farben sowohl in Text- als auch in Grafikdarstellungen ist in Kapitel 4.3 detailliert erläutert. Dort wird auch erklärt, wie der Monochromadapter auf Farben reagiert.

Modus	Art	Auflösung	Farben	Adapter	Bildschirm
0	Text	40 X 25	16 (grau)	FGA, EGA	Farbmonitor
1	Text	40 X 25	16 Vordergrund, 8 Hintergrund	FGA, EGA	Farbmonitor
2	Text	80 X 25	16 (grau)	FGA, EGA	Farbmonitor
3	Text	80 X 25	16 Vordergrund, 8 Hintergrund	FGA, EGA	Farbmonitor
4	Grafik	320 X 200	4	FGA, EGA	Farbmonitor
5	Grafik	320 X 200	4 (grau)	FGA, EGA	Farbmonitor
6	Grafik	640 X 200	2	FGA, EGA	Farbmonitor
7	Text	80 X 25	s/w	EGA, MA	S/W-Monitor
11	Intern im EGA				
12	Intern im EGA				
13	Grafik	320 X 200	16	EGA	Farbmonitor
14	Grafik	640 X 200	16	EGA	Farbmonitor
15	Grafik	640 X 350	s/w	EGA	Farbmonitor
16	Grafik	640 X 350	64	EGA	Farbmonitor

Abkürzungen: FGA: Farb-/Grafikadapter
EGA: Enhanced Graphics Adapter (HR-Farbgrafikadapter)
MA: Monochromadapter

Tabelle 4-1 Formateigenschaften der 15 Bildschirmmodi

4.2.1 Bildschirmauflösung

Grafikbilder werden aus einzelnen Punkten (*Bildelemente* oder *Pixel*) aufgebaut. Die Bildschirmauflösung ist durch Rasterzeilen und Pixelspalten definiert. Wieviel Rasterzeilen ein Monitor anzeigen kann, wird von der Hardware und den Bildschirmsignalen bestimmt, wir haben darüber softwareseitig so gut wie keine Kontrolle. Ein Bildschirm des Standard-PC verfügt über 25 Textzeilen (Zeichenzeilen) und 200 Grafikzeilen (Rasterzeilen). Um die Auflösung des Bilschirms zu verändern, muß die Anzahl der Pixel pro Rasterzeile geändert werden.
Die Grafikmodi des PC verfügen über drei Auflösungen: niedrig, mittel und hoch, mit 160, 320 oder 640 Pixel pro Zeile. Weder der Farb-/Grafikadapter noch der HR-Farbgrafikadapter (EGA) stellen die niedrige

Auflösung (160 x 200 Pixel) bereit (diese wird nur beim PCjr verwendet). Da Textzeichen auch im Grafikmodus dargestellt werden können, gibt es für die mittel- und hochauflösenden Modi die entsprechenden Textmodi. Ein schmaleres Zeichen, das im 80 Spalten/25 Zeilen-Format darstellbar ist, geht von einer Auflösung von 640 x 200 Pixel aus, ein breiteres Zeichen des Formates 40 Spalten/25 Zeilen basiert auf 320 x 200 Pixel. Daß die 80 x 25-Zeichendarstellung des Monochromadapters einen besser lesbaren Text erzeugt, liegt an der höheren Auflösungsbasis von 720 x 350 Pixel.

Auflösung	Pixel	Zeichen
Niedrig	160 X 200	20 X 25
Mittel	320 X 200	40 X 25
Hoch	640 X 200	80 X 25

Tabelle 4-2 Die Auflösung von Textzeichen im Grafikmodus

Niedrig-auflösende Grafiken haben ein einheitliches 20-Spalten-Text-Format, zu dem kein Äquivalent in den Standard-Textmodi existiert.

4.2.2 Steuerung des Bildschirmmodus

Der Bildschirmmodus wird vom ROM-BIOS mit Hilfe des Interrupt 16 (hex 10), das ist Routine 0, kontrolliert. BASIC ermöglicht einen uneingeschränkten Zugriff über das SCREEN-Kommando, benutzt aber eine eigene Erkennungsmethodik. Der BASIC SCREEN-Modus verwendet andere Zahlen für die einzelnen Modi, als das ROM-BIOS. Einige der Bildschirmmodi können auch von DOS aus gesteuert werden, jedoch befindet sich DOS in der Befehlsebene immer in einem Textmodus; es gibt hier keine Befehle, mit denen ein Grafikmodus aufgerufen werden könnte.

Modus	BASIC-Befehl	DOS-Befehl
0	SCREEN 0,0 : WIDTH 40	MODE BW40
1	SCREEN 0,1 : WIDTH 40	MODE CO40
2	SCREEN 0,0 : WIDTH 80	MODE BW80
3	SCREEN 0,1 : WIDTH 80	MODE CO80
4	SCREEN 1,0 oder SCREEN 4	–
5	SCREEN 1,1	–
6	SCREEN 2	–
7	–	MODE MONO

Tabelle 4-3 BASIC- und DOS-Befehle zum Ändern der Bildschirmmodi

4.3 Farben

Es stehen in jedem Modus, außer dem Monochrommodus, eine Reihe verschiedener Farben zur Verfügung. Wie Sie vielleicht schon bemerkt haben, ist die Anzahl der anwählbaren Farben von Modus zu Modus unterschiedlich. Im folgenden Abschnitt wollen wir die Farbmöglichkeiten der einzelnen Modi besprechen.
Farben werden auf den PC-Bildschirmen durch die Verknüpfung von vier Elementen erzeugt. Dies sind die drei Farbelemente rot, grün und blau sowie eine Intensitäts- oder Helligkeitskomponente. Text- und Grafikmodi verwenden dieselben Elemente zur Farberzeugung, kombinieren sie aber unterschiedlich. Die Textmodi, deren Basiseinheit ein Zeichen, aufgebaut aus einer Anzahl von Pixel, ist, benutzen ein ganzes Byte, um Farbe, Intensität und Blinkeigenschaft des Zeichens und seines Hintergrundes zu bestimmen. Die Grafikmodi, die eine kleinere Basiseinheit, nämlich ein Pixel, verwenden, brauchen nur zwischen einem und vier Bit, um Farbe und Helligkeit zu setzen; Blinkeigenschaft und der Hintergrund fallen weg. Das Setzen der Attribute für Text- und Grafikmodi behandeln wir eingehender im Kapitel 4.3.2 ff. Zuerst ein Wort zu den Farben selbst.
Die Farbnummern (0 bis 15), die von BASIC verwendet werden und grundsätzlich zur Festlegung der PC-Farben dienen, können hergeleitet werden, indem die vier Farbelemente als Bits einer Binärzahl angesehen werden. Wird ein 16-Farbenmodus verwendet, erhalten wir alle Farben von 0 bis 15, bei einem 8-Farbenmodus die Farben 0 bis 7 (das sind alle Farben ohne die Intensität). Mit einem 4-Farbenmodus bekommen wir eine Auswahl der 16 Gesamtfarben. Diese Auswahl wird *Palette* genannt. Im 2-Farbenmodus erhalten wir die Farben 0 und 7, schwarz und normales weiß (bzw. grün oder bernsteinfarben bei entsprechenden Monitoren).
So weit also die 16-Farben-Palette des Standard-PC, die sich aus den drei RGB-Farben (rot, grün, blau) und ihrer Intensität (I) zusammensetzt. Manchmal wird dies auch als *Grundfarbenschema IRGB* bezeichnet. Für den IBM PC wurde auch eine 64-Farben-Palette entwickelt, die aber nur auf der Kombination HR-Farbgrafikadapter (EGA) mit dem *Enhanced Color Display* oder ECD-Monitor, der *EGA/ECD-Kombination*, lauffähig ist. Diese ausgebaute Farbpalette rekrutiert sich ebenfalls aus den Farben rot, grün und blau, verfügt aber für jede Farbe über zwei unabhängige Signale, die eine hellere und eine dunklere Version der Farbe erzeugen können. Die Notation der 64-Farben-Palette lautet "RrGrBb", wobei die Großbuchstaben für die dunklere Version stehen. Beachten Sie, daß wir hier nicht über zwei Intensitätssignale, sondern über zwei vollkommen voneinander unabhängige Farbsignale sprechen, die für jede der drei Farben vier verschiedene Intensitäten zulassen. Für die Farbe rot würden die vier Intensitätsstufen "Rr" (stärkste Rotfärbung), "R.", ".r" und ".." (kein rot) lauten.

I	R	G	B	Zahl	Beschreibung
0	0	0	0	0	Schwarz
0	0	0	1	1	Blau
0	0	1	0	2	Grün
0	0	1	1	3	Türkis
0	1	0	0	4	Rot
0	1	0	1	5	Magenta
0	1	1	0	6	Braun (oder Dunkelgelb)
0	1	1	1	7	Hellgrau (oder normales Weiß)
1	0	0	0	8	Dunkelgrau (auf vielen Monitoren: Schwarz)
1	0	0	1	9	Hellblau
1	0	1	0	10	Hellgrün
1	0	1	1	11	Helltürkis
1	1	0	0	12	Hellrot
1	1	0	1	13	Hellmagenta
1	1	1	0	14	Gelb (oder Hellgelb)
1	1	1	1	15	Helles Weiß

Erklärung der Abkürzungen:
I – Intensitäts-Bit
R – Rot
G – Grün
B – Blau

Tabelle 4-4 Die volle Farbpalette des PC und die zugehörige 4-bit-Kodierung

Die 64-Farben-Palette der EGA/ECD-Kombination wird hier nicht weiter behandelt, weil sie nur selten benutzt wird, sehr speziell ist und kaum als standardmäßige Erweiterung anzusehen ist. Wir können in diesem Buch schließlich nicht alle exotischen Anschlußmöglichkeiten besprechen, um den Umfang des Buches nicht zu sprengen. Eine noch speziellere Adapter-Bildschirm-Kombination, der *IBM Professional Graphics Adapter* mit Monitor, verfügt über eine Farbpalette von 256 Farben und eine bemerkenswert hohe Auflösung. Doch auch eine Diskussion darüber würde uns zu weit führen, wir wenden uns deshalb wieder der Standard-Farbpalette zu.

Es gibt einige wichtige Dinge, die Sie bei der Wahl Ihrer Farben beachten sollten. Die vier Farbelemente (IRGB) produzieren alle aktiv Licht. Je mehr dieser Elemente Sie verwenden, desto heller wird die resultierende Farbe, sie wird aber auch "ausgewaschener". Für unser Auge sind die reinen Farben, wie rot, grün und blau, intensiver als die Mischfarben, wie türkis, magenta oder gelb. Dies gilt auch für die "intensiveren" (aufgehellten) Versionen der reinen Farben. Sie sollten aber noch drei andere Faktoren berücksichtigen:

1. Einige Farbmonitore sprechen nicht auf das Intensitäts-Bit an, was zur Folge hat, daß Farbe 8 genauso aussieht wie Farbe 0, Farbe 9 wie Farbe 1 usw.

2. Wird ein Monochrombildschirm mit einem Farb-/Grafikadapter verwendet, dann produzieren andere Farben als schwarz (0) und weiß (7) u.U. Fehlinformationen.
3. Programme, die auf dem PC oder XT mit dem Monochromadapter verwendet werden, müssen der ungewöhnlichen Behandlung von Farben auf dem Monochrombildschirm Rechnung tragen.
Über die Benutzung von Farben finden Sie in diesem Kapitel noch eine Menge interessanter und wichtiger Punkte, die Sie unbedingt lesen sollten, bevor Sie zur Praxis übergehen.

4.3.1 Farbunterdrückende Modi

Um die Grafikmodi kompatibel zu den vielen angebotenen Farb- und Monochromadaptern zu machen, fügte IBM einige Modi hinzu, die keine Farben produzieren, die sog. *farbunterdrückenden Modi*. Hiervon gibt es drei, es sind die Modi 0, 2 und 5. In diesen Modi werden alle Farben in Grauschattierungen verwandelt bzw. in Abstufungen derjenigen Farbe, die der Bildschirmbelag widergibt (etwa grün oder bernsteinfarben). Der Modus 5 kann genau vier Grautöne, die anderen beiden Modi können eine Vielzahl von Grauabstufungen erzeugen. Beachten Sie: Lediglich im Mischsignal wird die Farbe unterdrückt, nicht aber im RGB-Signal des Bildschirmadapters. Diese Inkonsequenz resultiert aus einem unvermeidbaren technischen Umstand.

Hinweis: Jeder farbunterdrückende Modus steht einem entsprechenden Farbmodus gegenüber. Die Modi 0 und 1 beziehen sich auf den 40-Spalten-Text, die Modi 2 und 3 auf den 80-Spalten-Text und die Modi 4 und 5 auf die mittelauflösende Grafik. Die Tatsache, daß die Modi 4 und 5 das Muster der Modi 0 und 1 und der Modi 2 und 3, bei denen der farbunterdrückende Modus zuerst kommt, umkehren, hat zu großer Verwirrung in BASIC beigetragen. Der Burst-Parameter des BASIC Kommandos SCREEN kontrolliert die Farbe. Die Bedeutung dieses Parameters wird von den Modi 4 und 5 umgekehrt, so daß der Befehl SCREEN,1 die Farbe in den Textmodi (0, 1, 2 und 3) aktiviert, in den Grafikmodi (4 und 5) aber unterdrückt. Diese Ungereimtheit ist wohl ursprünglich auf einen Programmierfehler zurückzuführen, heute jedoch ist sie Bestandteil der offiziellen Definition des SCREEN-Befehls. Die Tabelle in diesem Buch zeigt die korrekte Syntax für die Modi 0 bis 5.

Modus	Farbunterdrückung	Farbe aktiv
0	SCREEN 0,0 : WIDTH 40	
1		SCREEN 0,1
2	SCREEN 0,0 : WIDTH 80	
3		SCREEN 0,1 : WIDTH 80
4		SCREEN 1,0
5	SCREEN 1,1	

Tabelle 4-5 Die Farb-Burst-Parameter der Modi 0 bis 5. Die Modi 0 bis 3 folgen einer anderen Methode als die Modi 4 und 5.

4.3.2 Farbe in Text- und Grafikmodus

Sie sollten sehr gut über die Unterschiede zwischen Text- und Grafikmodus bezüglich der Farbe Bescheid wissen, insbesondere über die Ungereimtheiten, die bei der Benutzung von Textfarben auftreten. Im Textmodus haben Sie eine vollkommen eigenständige Kontrolle über die Farbe jeder einzelnen Zeichenposition. Für den Vordergrund steht die volle Farbpalette (16 Farben) zur Verfügung, der Hintergrund kann mit der 8-Farben-Palette aufgefüllt werden. Im Grafikmodus haben Sie die vollständige Kontrolle über die Farbe jedes Pixels und über die Farbe einzelner Operationen zur Erstellung der Grafikelemente, wie sie beispielsweise von BASIC bereitgestellt werden.
Theoretisch sollten uns die Grafikmodi ein reichhaltigeres Farbangebot für den gesamten Bildschirm bereitstellen. Schreiben wir aber einen Text in einem Grafikmodus, so haben wir keinerlei Kontrolle über die Hintergrundfarbe. Vielmehr ist diese immer die des gesamten Bildschirmes. Lesen Sie auch die Erläuterungen zum Palettenwert 0 des 4-Farbenmodus in Kapitel 4.3.2.3. Die Grafikmodi, die prinzipiell die bessere Farbkontrolle bieten, schneiden im Vergleich mit den Textmodi bei der Anzeige von Text also schlechter ab. Mit dieser Eigenart der Grafikmodi muß man leben.

4.3.2.1 Setzen der Farben in die Textmodi

In den Textmodi wird jede Zeichenposition auf dem Bildschirm von zwei zusammenhängenden Bytes kontrolliert. In Kapitel 3.2.2 finden Sie weitere Informationen über die Anordnung dieser Bytes im Speicher. Das erste Byte enthält den ASCII-Code des anzuzeigenden Zeichens (Anhang C listet alle Zeichen auf), das zweite Byte zeigt, wie das Zeichen erscheinen wird, spezifiziert seine Farbe usw. Dieses zweite Byte wird auch *Zeichenattribut* genannt.
Bevor wir uns tiefer in die Materie wagen, sollten wir erst noch zwei Begriffe klären, die meist nicht ganz richtig verwendet werden. In der IBM-PC-Terminologie über den Bildschirm sind beide Begriffe austauschbar. Es handelt sich um die Worte *Farbe* und *Attribut*. Beides sind technisch fest definierte Ausdrücke, die zwar ähnlich sind, sich aber doch unterscheiden. Oft findet man beide Begriffe praktisch synonym verwendet. Um unnötige Verwirrung zu vermeiden, sollten Sie beide als etwas vage Begriffe auffassen, die sich auf die Art, wie Dinge auf dem Bildschirm erscheinen, ebenso beziehen, wie auf die Datenkodierung im Speicher, die das Erscheinungsbild des Zeichens bestimmen.
Es gibt drei Komponenten des Textzeichenattributs: die Vordergrundfarbe (die Farbe des Zeichens), die Hintergrundfarbe (die Farbe des Feldes, das das Zeichen nicht einnimmt) und die Blinkkomponente des Zeichens. Der

Vordergrund kann alle 16 Farben der PC-Palette benutzen, der Hintergrund nur die acht Farben 0 bis 7, also die Grundfarben ohne Hinzunahme der Intensität.
Jede Zeichenposition auf dem Bildschirm hat ihre eigene Attributkontrolle, die unabhängig von allen anderen Bildschirmzeichen gesetzt wird. Jedes der acht Bits im Attribut-Byte kontrolliert selbständig ein Element des Anzeigeattributs. Wird kein Attribut angegeben, so setzen DOS und BASIC von sich aus hex 07, das ist weiß (7) als Vordergrund auf schwarz (0) als Hintergrund ohne Blinken.

Bit 7	6	5	4	3	2	1	0	Verwendung
1	.	.	.	.	.	.	.	Blinken des Vordergrundes
.	1	.	.	.	.	.	.	Rot-Komponente der Hintergrundfarbe
.	.	1	.	.	.	.	.	Grün-Komponente der Hintergrundfarbe
.	.	.	1	.	.	.	.	Blau-Komponente der Hintergrundfarbe
.	.	.	.	1	.	.	.	Intensitäts-Komponente der Vordergrundfarbe
.	.	.	.	.	1	.	.	Rot-Komponente der Vordergrundfarbe
.	.	.	.	.	.	1	.	Grün-Komponente der Vordergrundfarbe
.	.	.	.	.	.	.	1	Blau-Komponente der Vordergrundfarbe

Tabelle 4-6 Die Kodierung des Farbattribut-Byte

Die Farbqualität ist stark vom Monitor abhängig. Bei vielen Farbmonitoren sind die helleren, intensiveren Farben von einem Hintergrund gleicher Farbe ohne den Intensitäts-Zusatz gut unterscheidbar. Andererseits ignorieren manche Monitore das Setzen des Intensitäts-Bits, wodurch manche Vorder- und Hintergrundfarbkombinationen auf diesen Geräten unlesbar werden, z.B. gelb auf braun.
Bei der Darstellung ganzer Zeichen spricht einiges für den Einsatz eines Textmodus gegenüber einem Grafikmodus. Der größte Vorteil besteht darin, daß der Textmodus Zeichen schneller anzeigen kann, da er die Zeichen einer bereitgestellten Tabelle entnimmt, während ein Grafikzeichen hingegen Bit für Bit aus dem Speicher auf den Bildschirm übertragen werden muß. Die Textmodi benötigen weniger Speicherplatz und haben daher noch genügend Speicherreserve, um mehrere Textseiten direkt im Bildschirmspeicher abzulegen und sie einzeln in schneller Reihenfolge aufzurufen. Auch gibt es in den Textmodi einige Spezialeffekte, die von den Grafikmodi nicht bereitgestellt werden können, wobei an erster Stelle die größere Farbauswahl und das Blinken eines Zeichens zu nennen wären.

4.3.2.2 Setzen der Attribute im Monochrommodus

Der Monochrommodus (Modus 7), der vom IBM-Monochromadapter verwendet wird, verfügt über eine beschränkte Auswahl von Darstellungseigenschaften, die als Ausgleich für die fehlenden Farben angesehen werden können. Für das Setzen der Anzeigeattribute für Monochromzeichen wird dasselbe Schema benutzt, das auch bei Textmoduszeichen in den Grafikmodi 0 bis 3 Anwendung findet.
Im Monochrommodus werden die Blink- und Intensitäts-Bits verwendet. Dennoch bringen nur vier "Farb"-Kombinationen von Vorder- und Hintergrund unterscheidbare Ergebnisse:
1. Normal weiß auf schwarz. Dies wird erzeugt, indem weiß (Vordergrund-Bits 111) auf schwarz (Hintergrund-Bits 000), oder hex 07, ausgewählt wird.
2. Unterstrichene Zeichen. Diese werden generiert, indem dem Attribut-Byte der Wert hex 01 zugewiesen wird, was blau (Vordergrund-Bits 001) auf schwarz (Hintergrund-Bits 000) entspricht.
3. Umgekehrte Darstellung (invers oder revers) wird durch schwarz (Vordergrund-Bits 000) auf weiß (Hintergrund-Bits 111) oder hex 70, erreicht.
4. Unsichtbare-Zeichen. Diese werden erzeugt, indem schwarz (Vordergrund-Bits 000) auf schwarz (Hintergrund-Bits 000) oder hex 00 gewählt wird.
Alle anderen "Farb"-Kombinationen zeigen den gleichen Effekt, wie normales weiß auf schwarz (hex 07). Andere logisch erscheinende Kombinationen, wie unsichtbare Weiß-auf-weiß- oder Inversdarstellungen mit gleichzeitiger Unterstreichung existieren nicht im Monochrommodus. Beachten Sie aber, daß die Blink- und Intensitäts-Attribute von den obigen Kombinationen unabhängig sind.

4.3.2.3 Setzen der Farben in die Grafikmodi

Wir wissen jetzt, wie man Farben in den Textmodi bzw. die Äquivalente für Farben im Monochrommodus setzt. Die Darstellung von Farben im Grafikmodus erfolgt nach einem anderen Prinzip. In den Grafikmodi (Modi 4 bis 6 und 8 bis 10) korrespondiert jedes Pixel des Bildschirms mit einer Farbe. Die Farbe wird genauso gesetzt, wie die Attribute im Textmodus, es gibt aber wichtige Unterschiede. Erstens können Pixel nicht blinken und zweitens ist jedes Pixel ein einzelner (diskreter) für sich stehender Farbpunkt. Es gibt keinen Vorder- und Hintergrund, jeder Pixel besteht aus der einen oder anderen Farbe. Beim Schreiben von Text wird eine Farbe für die Pixel, die den Hintergrund darstellen und eine Farbe für die Pixel der Zeichen, also des Vordergrundes, gewählt.

Hinweis: Die Benutzung der Grafikmodi in BASIC erweckt den Eindruck, daß es eine Hintergrundfarbe für Grafiken gäbe. Das beruht auf folgender Festlegung, die BASIC übernommen hat: Jeder Pixel, der nicht explizit mit einer Vordergrundfarbe gekennzeichnet ist, erhält die Hintergrundfarbe zugeordnet. Die ROM-BIOS-Text-Routine hält sich ebenfalls an diese Festlegung. Mehr darüber finden Sie in Kapitel 12.

Für jeden Grafikmodus gibt es eine vordefinierte Auswahl an Farben, eine Palette. Die Standardpalette des HR-Farbgrafikadapters (EGA) kann verändert werden, nicht jedoch die des Original-Farb-/Grafikadapters. Ist erst einmal eine Farbpalette für jeden Grafikmodus festgelegt, kann die Farbe jedes Pixel mit Hilfe des Farbwertes in den dem Pixel zugeordneten Bits bestimmt werden. In einem 2-Farbenmodus gibt es für jedes Pixel ein Bit, das je nach Farbe gleich 0 oder 1 ist. In einem 4-Farbenmodus existieren zwei Bits mit den Werten 0 bis 3, und in einem 16-Farbenmodus gehören zu jedem Pixel vier Bits, deren Inhalt von 0 bis 15 reicht. Die Farbwerte, die zur Definition eines Pixel verwendet werden, müssen nicht notwendigerweise dieselben Zahlen sein, die der Identifikation der auf dem Bildschirm abgebildeten Farben dienen (0 bis 15).

Bits	Wert	Farbe
0 0	0	0 Schwarz
0 1	1	7 Weiß

Tabelle 4-7 Die Standard-Palette für den 2-Farben-Grafikmodus (Modus 6)

Im 2-Farbenmodus 6 gibt es nur eine Standard-Farbpalette. In den 4-Farbenmodi 4 und 5 gibt es zwei Standard-Paletten: Palette 0 und Palette 1. Zwei Anmerkungen sind zu diesen Paletten zu machen: Erstens kann der Palettenwert 0 von Schwarz (Farbe 0) auf jede andere Farbe verändert werden. Zweitens ist der Palettenwert 0 stets die Hintergrundfarbe und Palettenwert 3 immer die Vordergrundfarbe beim Schreiben von Textzeichen. Im 4-Farbenmodus 10 gibt es nur eine Standard-Palette, die mit der Palette 1 übereinstimmt. Die 16-Farbenmodi 13 und 14 (sowie 8 und 9 des PCjr) haben eine Standard-Palette, die, wie Sie vielleicht schon erwartet haben, mit den aktuellen Farbnummern von 0 bis 15 übereinstimmen. Die Farbmodi 13 und 14 können nur mit dem HR-Farbgrafikadapter (EGA) verwendet werden.

Bits	Wert	Farbe
0 0	0	0 Schwarz (Standardwert, kann auf jede Farbe geändert werden)
0 1	1	2 Grün
1 0	2	4 Rot
1 1	3	6 Braun

Tabelle 4-8 Palette 0, eine der zwei Standardpaletten der 4-Farben-Grafikmodi (Modi 4 und 5)

Bits	Wert	Farbe
0 0	0	0 Schwarz (Standardwert, kann auf jede Farbe geändert werden)
0 1	1	3 Türkis
1 0	2	5 Magenta
1 1	3	7 Normal Weiß

Tabelle 4-9 Palette 1, eine der zwei Standardpaletten der 4-Farben-Grafikmodi (Modi 4 und 5)

4.3.2.4 Veränderbare Farbpaletten des HR-Farbgrafikadapters (EGA)

Bis jetzt haben wir nur über die Standardfarben, die mit Hilfe der Standardpaletten erzeugt werden, gesprochen. Die Farbpalette des Original-Farb-/Grafikadapter ist festgelegt und kann nicht geändert werden. Bei anderen Adaptern besteht unter Umständen eine solche Möglichkeit, z.B. beim HR-Farbgrafikadapter (EGA), dessen Farben innerhalb der Paletten ausgewechselt werden können. Für das Wechseln einer Farbe wird einfach der Farbwert einer anderen Farbe zugeteilt. Die Anforderung des Wertes für Farbe 1 (blau) kann dann beispielsweise die Farbe 4 (rot) erzeugen.
Die Zuordnung eines Palettenwertes zu einem aktuellen Farbwert untersteht der Kontrolle der gerade aktuellen Palette. Die Paletten können in BASIC mit den Palettenbefehlen oder mit der Bildschirmroutine des BIOS geändert werden.

4.4 Aufbau des Bildschirmspeichers

Wir wenden uns dem internen Aufbau des Bildschirmspeichers zu. Dabei werden Sie wichtige Informationen über den Zusammenhang zwischen Bildschirmspeicher und Bildschirm kennenlernen.
Bei der Benutzung der Bildschirmmodi, die ihren Bildschirmspeicher im Block B untergebracht haben (dies sind die Farb-/Grafikmodi 0 bis 6 und der Monochrommodus 7), ist es für Programme ungefährlich, im Bildschirmspeicher Veränderungen vorzunehmen. IBM hielt von dieser Idee ursprünglich nicht viel, hat sich aber den Gegebenheiten angepaßt und versucht in allen derzeitigen und zukünftigen Bildschirmadaptern, den direkten Zugriff zu unterstützen. Für die erweiterten Modi des HR-Farbgrafikadapters (Modi 13 bis 16) ermöglichte IBM eine leichte Benutzung des Bildschirmspeichers. Der Bildschirmspeicher des HR-Farbgrafikadapters (EGA) ist theoretisch im A-Block anzutreffen, Programme können den Speicher an dieser Stelle jedoch nicht ansprechen. Diese Barriere sollte nicht durchbrochen werden.
Die Einsatzmöglichkeiten und die Kodierungen des Bildschirmspeichers variieren mit dem sich gerade in Betrieb befindlichen Modus. Die Modi 0

bis 6 gehören dem Farb-/Grafikadapter an, der Modus 7 dem Monochromadapter. Die Modi 11 bis 16 können nur mit dem HR-Farbgrafikadapter (EGA) verwendet werden.
Der Bildschirmspeicher der Modi 0 bis 6 belegt 16 Kbyte, der des Monochrommodus nur 4 Kbyte. Die Textmodi, sowohl des Farb-/Grafikadapters als auch des Monochromadapters, benötigen weniger Speicherkapazität als die Grafikmodi, denn sie belegen nur zwei Bytes pro Zeichen (siehe Kapitel 4.4.3). Eine 80/25-Zeichen-Textdarstellung kommt mit einem Speicherplatz von 4000 Bytes aus. Eine Grafikdarstellung kann zwischen 16 und 32 Kbyte Speicher benötigen, je nachdem, wie viele Farben verwendet werden. In den 2-Farben-Grafikmodi verbraucht jedes Pixel ein Bit, in den 4- und 16-Farbenmodi benötigt jedes Pixel zwischen zwei und vier Bits, um den größeren Farbwert binär speichern zu können. Eine 16-farbige Darstellung mit einer Auflösung von 320 mal 200 Pixel verbraucht volle 32 Kbyte Speicherplatz, nämlich ein Byte für vier Pixel.
Eine typische Textdarstellung benötigt also 4000 Bytes Speicherplatz (mit einer Bildschirmdarstellung von 40 Spalten nur 2000 Bytes) Es steht somit im 16-Kbyte-Bildschirmspeicher des Farb-/Grafikadapters noch eine Menge freier Platz zur Verfügung. Dieser freie Speicherbereich kann für umfangreichere Texte verwendet werden, die man in Bildschirmseiten aufteilt.

Modus	Mindestens erforderlicher Speicher	Segmentadresse (hex)	Adapterkarte
0	2 Kbyte	B800	Farb-/Grafik
1	2 Kbyte	B800	Farb-/Grafik
2	4 Kbyte	B800	Farb-/Grafik
3	4 Kbyte	B800	Farb-/Grafik
4	16 Kbyte	B800	Farb-/Grafik
5	16 Kbyte	B800	Farb-/Grafik
6	16 Kbyte	B800	Farb-/Grafik
7	4 Kbyte	B000	Monochrom
13	32 Kbyte	A800	HR-Farbgrafik (EGA)
14	32 Kbyte	A800	HR-Farbgrafik (EGA)
15	64 Kbyte	A000	HR-Farbgrafik (EGA)
16	32 Kbyte	A800	HR-Farbgrafik (EGA)

Tabelle 4-10 Speicherbereich, den jeder Modus benötigt und die jeweilige Startposition des Speichers

4.4.1 Handhabung mehrerer Seiten in den Textmodi

Die Textmodi 0 bis 3 benötigen nur einen Bruchteil der ihnen zur Verfügung stehenden Speicherkapazität von 16 Kbyte. Die Modi 0 und 1 belegen nur 2 Kbyte, die Modi 2 und 3 jeweils 4 Kbyte. Für diese Modi wird der bereitstehende Speicherplatz in mehrere Bildschirmabbilder geteilt, die

Bildschirmseiten oder einfach nur *Seiten* genannt werden. Zu jedem Zeitpunkt wird eine dieser Seiten dargestellt. Informationen können sowohl in die angezeigte als auch in momentan nicht sichtbare Seiten geschrieben werden. Mit Hilfe dieser Technik läßt sich eine Bildschirmdarstellung aufbauen, während eine andere angezeigt wird; zu gegebener Zeit kann dann auf das vorbereitete Bild umgeschaltet werden. Dieses Umschalten vermittelt den Eindruck, daß der Bildschirm von einer auf die andere Sekunde komplett wechselt.
Die Seitenzahlen in den Modi 0 und 1 gehen von 0 bis 7, in den Modi 2 und 3 von 0 bis 3, wobei Seite 0 am Anfang des 16-Kbyte-Speicherbereichs liegt. Jede Seite beginnt an einer geradzahligen Offsetadresse als untere Speichergrenze, also bei 0 Kbyte, 2 Kbyte, 4 Kbyte, etc.

Seite	Modi 0 und 1 2 Kbyte Aufteilung	Modi 2 und 3 4 Kbyte Aufteilung
0	B800	B800
1	B880	B900
2	B900	BA00
3	B980	BB00
4	BA00	
5	BA80	
6	BB00	
7	BB80	

Tabelle 4-11 Die Offsetadressen der Bildschirmseiten in den Modi 0 bis 3

Die Seiten werden ausgewählt, indem die Startadresse geändert wird, die im 6845-Chip Verwendung findet. Normalerweise geschieht dies mit Hilfe der ROM-BIOS-Routine 5 (Interrupt 16 (hex 10)). Diese Routine ist in Kapitel 9 näher erläutert.
In jedem dieser Modi kann der Speicherplatz, der gerade nicht aktiv genutzt wird, d.h., nicht auf dem Bildschirm erscheint, für andere Zwecke eingesetzt werden. Es ist allerding kaum ratsam, von dieser Möglichkeit Gebrauch zu machen, da man sich mit dieser Methode sehr schnell Ärger einhandelt, wenn die Verwaltung nicht wirklich exakt funktioniert.

4.4.2 Seitendarstellung in den Grafikmodi

Im HR-Farbgrafikadapter (EGA) und in anderen Adaptern, die über genügend Speicherplatz verfügen, ist das Konzept der Bildschirmseiten auch für die Grafikmodi vorhanden. Der Hauptvorteil des Seitenumschaltens in den Grafikmodi besteht darin, die Informationen auf dem Schirm schnell austauschen zu können. In einem Grafikmodus lassen sich auf diese Weise - zumindest theoretisch - stetig fließende Animationseffekte erreichen.

Die Animation kann aber leider nicht sehr weit getrieben werden, da der dafür nötige Speicherplatz nicht zur Verfügung steht. Die Benutzung von Bildschirmseiten in den Grafikmodi ist praktisch in allen neueren Adapterkarten grundsätzlich vorgesehen.

4.4.3 Zeichendarstellung in den Text- und Grafikmodi

Wie Sie nunmehr wissen, werden in den Textmodi (sowohl beim Monochrom- als auch beim Farb-/Grafikadapter) keine Zeichenabbilder gespeichert, sondern nur der ASCII-Wert des Zeichens und seine Anzeigeattribute. Das Zeichen wird von einem Zeichengenerator auf den Bildschirm gebracht, der Teil des Adapters ist. Der Zeichengenerator des Farb-/Grafikadapters erzeugt seine Zeichen in einem 8-mal-8-Pixel-Blockformat, während der Monochromadapter mit einem Zeichengenerator des Blockformates mit 9 mal 14 Pixel arbeitet. Das größere Zeichenformat ist einer der Faktoren, die für die bessere Lesbarkeit der Monochromdarstellung verantwortlich zeichnen.
Die Standard-ASCII-Zeichen von CHR$(1) bis CHR$(127) sind nur die Hälfte der zur Verfügung stehenden Zeichen in den Textmodi. Weitere 128 Grafikzeichen werden durch denselben Zeichengenerator bereitgestellt (CHR$(128) bis CHR$(255)). Über die Hälfte davon kann für das Entwerfen einfacher Strichzeichnungen benutzt werden. Eine komplette Liste der darstellbaren Zeichen finden Sie in Anhang C.
Die Grafikmodi können auch Zeichen darstellen, die aber anders erzeugt werden. In den Grafikmodi werden Informationen stets Bit für Bit gespeichert. Die Zeichen machen hier keine Ausnahme, jedes Bit muß einzeln gesetzt werden. Der große Vorteil dieser Darstellungsart besteht darin, daß Zeichen leicht selbst entworfen werden können. Im Original-Farb-/Grafikadapter liegt die Tabelle für den zweiten Block von 128 Zeichen im RAM, läßt sich also verändern. Nachdem die Änderungen ausgeführt wurden, kann der Benutzer direkt auf einen anwendungsbezogenen und selbstdefinierten Zeichensatz zugreifen.

4.4.3.1 Speichern von Zeichen in den Textmodi

Die Speicherung des Bildschirms beginnt im Textmodus in der linken oberen Ecke und geht von links nach rechts und von oben nach unten, so wie wir auch Zeile für Zeile lesen.
Die Modi 0 und 1 besitzen ein Bildschirmformat von 40 Spalten und 25 Zeilen. Jede Zeile benötigt 80 Bytes (40 mal 2). Der gesamte Bildschirm erfordert 2 Kbyte Speicherbereich, die 16 Kbyte, die reserviert sind, können also acht Seiten aufnehmen. Sind die Zeilen von 0 bis 24 und die Spalten von 0 bis 39 numeriert, kann jeder Zeichenoffset in der ersten Seite nach folgender Formel berechnet werden:

```
Zeichenoffset = (Zeilennummer x 80) + (Spaltennummer x 2)
```

Das Attribut-Byte folgt immer dem Byte, das den ASCII-Zeichenwert enthält, seine Speicherposition kann also einfach durch Addition einer 1 zum Zeichenoffset errechnet werden.
Die Modi 2, 3 und 7 sind auch Textmodi, die aber mit 80 Spalten arbeiten. Das Byte-Muster ist dasselbe wie in den 40-Spalten-Modi, nur erfordert jede Zeile doppelt so viel Speicher, mithin also 160 Bytes (80 mal 2). Die Folge ist, daß das 80-Spalten/25-Zeilen-Bildschirmformat 4 Kbyte benötigt und der 16-Kbyte-Speicher daher nur vier Seiten aufnehmen kann. Der Offset jeder Bildschirmposition der ersten Seite läßt sich durch folgende Formel ausdrücken:

```
Zeichenoffset = (Zeilennummer x 160) + (Spaltennummer x 2)
```

Die Seiten des Farb-/Grafikadapters beginnen üblicherweise immer an einer geraden Kbyte-Grenze. Da jede Seite im Textmodus genau 2000 oder 4000 Bytes beansprucht, folgen jeder Seite einige ungenutzte Bytes (je nach Länge der Seite: 48 oder 96). Um eine Bildschirmposition in jeder beliebigen Seite zu erhalten, verwenden Sie einfach die folgende Formel:

```
Position = (Segmentadresse x 16) + (Seitennummer x Seitenlänge)
 + (Zeilennummer x Zeilenlänge x 2)
 + (Spaltennummer x 2) + Schalter
```

Erklärung der Begriffe:
Position ist die 20-bit-Adresse der Bildschirminformation.
Segmentadresse ist der Beginn des Speicherbereichs im Bildschirmspeicher (z.B. hex B000 oder hex B800).
Seitennummer liegt zwischen 0 und 3 oder 0 und 7.
Seitenlänge ist gleich 2 Kbyte oder 4 Kbyte .
Zeilennummer liegt zwischen 0 und 24.
Zeilenlänge ist 40 oder 80.
Spaltennummer liegt zwischen 0 und 39 oder 0 und 79.
Schalter ist 0 für das Zeichen selbst und 1 für das Attribut.

4.4.3.2 Speichern von Pixel in den Grafikmodi

Pixel werden im Grafikmodus als Bit-Serien gespeichert, d.h., zu jedem Pixel auf dem Bildschirm existiert ein zugehöriges Bit im Speicher. Es gibt drei unterschiedliche Speicherschemata.
Der Original-Farb-/Grafikadapter organisiert den Bildschirm in 200 Zeilen von 0 bis 199. Die Anzahl der Pixel pro Zeile hängt vom gewählten Modus ab. Die Modi 4 und 5 haben eine mittlere Auflösung mit 320 Pixel pro Zeile, der Modus 6 ist hochauflösend mit 640 Pixel pro Zeile. Die Pixel für niedrige, mittlere und hohe Auflösung sind von 0 bis 159, 319 oder 639 durchnumeriert.

Der Speicher für die Zeilen ist in *Zeilenblöcke* unterteilt, die zusammenhängenden Speicherplatz beanspruchen. Für die Modi 4, 5 und 6 gibt es zwei Blöcke, einer speichert die geradzahligen Zeilen 0, 2, 4, usw. bis 198, der andere speichert die ungeradzahligen Zeilen 1, 3, 5, usw. bis 199.

Modus	Spalten	Farben	Bits	Blöcke	Speicher (Kbyte)
4	320	4	2	2	16
5	320	4	2	2	16
6	640	2	1	2	16

Tabelle 4-12 Die Formate und Speicherbelegungen der Grafikmodi

Der Speicherbereich, der jedem Pixel zugeordnet ist, variiert je nach Modus. Der Modus 6 braucht nur ein Bit pro Pixel, die Modi 4 und 5 benötigen zwei Bits, da die Auswahl zwischen vier Farben besteht. Die Modi, die 16 Farben zur Verfügung stellen, beanspruchen vier Bits pro Pixel. Die Bits für jedes Pixel jeder Zeile werden in aufeinanderfolgender Anordnung aus dem Speicher gelesen. Für den Modus 6, der ein Bit pro Pixel benötigt, sind die ersten acht Bits den ersten acht Pixel des Bildschirmes zugeordnet. Das erste Bit bestimmt das erste Pixel, und so weiter. Im Modus 4, der zwei Bits pro Pixel benötigt, legen die ersten acht Bits die ersten vier Pixel des Bildschirmes fest.
In den Modi 4, 5 und 6 benötigt jede Pixel-Zeile genau 80 Bytes.

4.5 Bildschirmsteuerung

Grundsätzlich kann die Bildschirmanzeige auf vier verschiedene Weisen gesteuert werden:

- Indem die Unterstützung der Programmiersprachen genutzt wird (z.B. das SCREEN-Kommando in BASIC).
- Indem die DOS-Routinen verwendet werden (siehe Kapitel 16 und 17).
- Indem die ROM-BIOS-Bildschirmroutinen benutzt werden (Kapitel 9).
- Durch direkte Hardwaremanipulation über Speicherzugriff und Ports.

Die Bildschirmroutinen, die von den Programmiersprachen, DOS und dem ROM-BIOS bereitgestellt werden, plazieren die Daten der Bildschirmausgabe automatisch in den dafür vorgesehenen Bildschirmspeicher, wobei jede Routinenart (BIOS, DOS, Programmiersprachen) andere Kontrollmöglichkeiten bietet. Die 16 ROM-BIOS-Routinen, die sich auf den Bildschirm beziehen, sind sehr umfassend. Mit ihnen läßt sich nahezu alles durchführen, um eine erwünschte Darstellung zu erhalten; man kann eine einfache Darstellung auf dem Bildschirm erzeugen, den Cursor steuern

Bit 7	6	5	4	3	2	1	0	Pixel	Bit 7	6	5	4	3	2	1	0	Pixel
Modus 6									Modi 4 und 5								
X	.	.	.	.	.	.	.	1	X	X	.	.	.	.	.	.	1
.	X	.	.	.	.	.	.	2	.	.	X	X	.	.	.	.	2
.	.	X	.	.	.	.	.	3	.	.	.	.	X	X	.	.	3
.	.	.	X	.	.	.	.	4	.	.	.	.	.	.	X	X	4
.	.	.	.	X	.	.	.	5									
.	.	.	.	.	X	.	.	6	Modi 13 und 14								
.	.	.	.	.	.	X	.	7	X	X	X	X	.	.	.	.	1
.	.	.	.	.	.	.	X	8	.	.	.	.	X	X	X	X	2

Tabelle 4-13 Speicherschemata der ersten Pixel in drei Grafikformaten

und Bildschirmdaten manipulieren. Einen Überblick entnehmen Sie bitte Kapitel 9. Im Rahmen einer Kontrolloptimierung haben wir auch die Möglichkeit, die Softwareebene zu umgehen und Informationen direkt in den Bildschirmspeicher einzugeben. Diese Methode sollte aber nur angewendet werden, wenn auch ein guter Grund dafür vorhanden ist.
Bevor Sie sich für die direkte Bildschirmausgabe entscheiden, sollten Sie wissen, daß Sie damit leicht mit Fenstersystemen (*Windows*) in Konflikt geraten, ebenso wie mit den erweiterten Multitasking-Betriebssystemen. Trotz alledem benutzen viele Programme diese Methode, man kann fast schon sagen, daß es in gewisser Weise ein Standard geworden ist. Auf lange Sicht muß von dieser Unart abgeraten werden, obwohl sich heute jedermann ihrer zu bedienen scheint.
Grundsätzlich kann man Programme, die eine direkte Bildschirmspeicherverwaltung durchführen, nicht mit Fenstersystemen zusammenarbeiten lassen, weil sie um die Kontrolle des gleichen Speicherbereiches kämpfen und gegenseitig die Daten überschreiben würden. IBMs "Topview" ist z.B. in der Lage, mit Programmen zu kooperieren, die direkt auf den Bildschirmspeicher zugreifen, das ist auch der Grund für "Topviews" enorme Länge. Die von "Topview" vorgenommene Anpassung besteht darin, daß vom Bildschirmspeicher eine Kopie angefertigt wird. Läuft ein Programm, so wird diese Kopie in den Speicher zurückgeladen und nach Beendigung der Vorgänge wird eine neue Kopie angefertigt. Diese Arbeitsweise erlaubt es dem Programm "Topview" mit anderen Programmen, die den Bildschirmspeicher verwenden, zusammenzuarbeiten. Natürlich müssen auch eine Menge handfester Nachteile in Kauf genommen werden: Erstens wird die Verarbeitungsgeschwindigkeit beeinträchtigt und der Speicherbedarf ist höher; zweitens kann das Programm im Hintergrund nicht simultan mit anderen Programmen ablaufen; und drittens lassen sich die Bildschirmdaten nicht "fensterln", d.h., sie können z.B. nicht zu einem Fenster transferiert werden.

Der Programmierer wird mit einem Konflikt konfrontiert: Die direkte Ausgabe auf den Bildschirm hat den Vorteil von Schnelligkeit und Effizienz, während das BIOS oder höhere Ebenen (z.B. DOS) den Vorteil größerer Flexibilität mit sich bringen und auch mit Fenstersystemen, neuen Adaptern usw. verwendet werden können. Eine Lösung, die als sehr angenehm empfunden werden kann, besteht darin, beide Vorgehensweisen in *ein* Programmsystem aufzunehmen und je nach Anforderung die eine oder die andere Variante zu aktivieren.

4.5.1 Direkte Hardwarekontrolle

Das Gros der in diesem Kapitel beschriebenen Informationen, vor allem über das interne Speicherkonzept, dienen in erster Linie dem direkten Programmieren, d.h., dem Schreiben von Informationen direkt in den Bildschirmspeicher. Sie sollten sich aber immer des Risikos bewußt sein, das die direkte Programmiertechnik mit sich bringt. Es empfiehlt sich grundsätzlich, stets auf der höchstmöglichen Ebene zu arbeiten, auf der eine für die jeweilige Anwendung ausreichende Kontrolle noch gegeben ist. Das Zugreifen auf niedere Ebenen ist im allgemeinen nicht ungefährlich, vor allem das direkte Programmieren, das nur in ganz wenigen Ausnahmen Sicherheit und Zuverlässigkeit bietet.

Monochromadapter: E/A-Ports
Der Monochromadapter verwendet vier Ports: die Bildschirm- und Statusports und die Register des 6845 Bildschirm-Controller.
Der Bildschirmkontrollport (hex 3B8): Drei der acht Bits dieses Ports können wir setzen: das Bit für hohe Auflösung (das beim Monochromadapter immer gesetzt sein muß), das Bildschirm-Bit und das Blink-Bit. Das Bildschirm- und Blink-Bit schalten die Bildschirmanzeige bzw. die Eigenschaft *Blinken* an und aus. Der Wert hex 29 setzt den Port auf seine normale Stellung.

Bits 7	6	5	4	3	2	1	0	Verwendung
.	.	.	.	.	.	.	X	Hochauflösender Modus, muß auf 1 sein
.	.	.	.	.	X	X	.	Unbenutzt
.	.	.	.	X	.	.	.	0 = Bildschirmsignal unterdrücken; 1 = Bildschirmsignal ermöglichen
.	.	.	X	.	.	.	.	Unbenutzt
.	.	X	.	.	.	.	.	0 = Blinkfunktion aus; 1 = Blinkfunktion an
X	X	.	.	.	.	.	.	Unbenutzt

Tabelle 4-14 Die Kodierung des Bildschirmkontrollports

Der Bildschirmstatusport (hex 3BA): Dieser Port speichert den Status des horizontalen Synchronisationssignals in Bit 0 und den bitweisen Bildschirmdatenstrom in Bit 3. Obwohl diese zwei Bits zugänglich sind, bringen sie keinen besonderen Nutzen. Die anderen Bits des Port sind unbenutzt.

Der 6845 Bildschirm-Controller (hex 3B4 und 3B5): Im 6845 gibt es 18 programmierbare interne Register, in denen solche Dinge wie die zeitliche Abstimmung des vertikalen und horizontalen Synchronisationssignals oder die Anzahl der Zeilen und die Anzahl der Zeichen pro Zeile spezifiziert werden. Nur vier der Register sind ohne Wagnis zu benutzen, die Register hex 0A, 0B, 0E und 0F. Die Register 0A und 0B bestimmen die Zeilen, in denen der Cursor beginnt und endet, die Register 0E und 0F spezifizieren die Bildschirmposition des Cursor, mit einem Wert zwischen 0 und 1999. Diese beiden Funktionen sind auch über den Interrupt 16 (hex 10) mit Hilfe der BIOS-Routinen zugänglich. Versuchen Sie nicht, mit anderen Werten zu spielen, denn dadurch können Hardwarekomponenten zerstört werden. Mehr hierzu finden Sie im Technischen Referenzhandbuch von IBM.

Farb-/Grafikadapter: E/A-Ports

Um die Grafikfunktionen alle unterzubringen, hat der Farb-/Grafikadapter mehr E/A-Ports als der Monochromadapter. Die wichtigsten Aspekte der sieben Ports finden Sie aufgelistet:

Das Modusauswahlregister (hex 3D8): Dieses Byte wird benutzt, um von einem Modus in einen anderen zu gelangen.

Bits 7	6	5	4	3	2	1	0	Verwendung
.	.	.	.	.	.	.	X	0 = 40 X 25 Textmodus wählen; 1 = 80 X 25 Textmodus wählen
.	.	.	.	.	.	X	.	0 = Textmodus wählen; 1 = 320 X 200 Grafikmodus wählen
.	.	.	.	.	X	.	.	0 = Farbmodus wählen; 1 = S/W-Modus wählen
.	.	.	.	X	.	.	.	0 = Bildschirmsignal unterdrücken; 1 = Bildschirmsignal ermöglichen
.	.	.	X	.	.	.	.	1 = 640 X 200 S/W-Grafik wählen
.	.	X	.	.	.	.	.	0 = Blinkfunktion aus; 1 = Blinkfunktion an
X	X	.	.	.	.	.	.	Unbenutzt

Tabelle 4-15 Die Kodierung des Modusauswahlregisters

Das Farbauswahlregister (hex 3D9): Dieses Byte wird benutzt, um die Randfarben des Bildschirms in den Textmodi und die Hinter- und Vordergrundfarben in den Grafikmodi zu verändern.

Bits 7	6	5	4	3	2	1	0	Verwendung
.	.	.	.	.	.	.	X	Blauen Vorder-, Hintergrund oder Rand wählen
.	.	.	.	.	.	X	.	Grünen Vorder-, Hintergrund oder Rand wählen
.	.	.	.	.	X	.	.	Roten Vorder-, Hintergrund oder Rand wählen
.	.	.	.	X	.	.	.	Intensität wählen
.	.	.	X	.	.	.	.	Alternative, intensivierte Palette wählen
.	.	X	.	.	.	.	.	0 = Palette 0; 1 = Palette 1
X	X	.	.	.	.	.	.	Unbenutzt

Tabelle 4-16 Die Kodierung des Farbauswahlregisters

Das Statusregister (hex 3DA): Dieses Register speichert wertvolle Informationen für denjenigen, der flimmer- und schneefreie Bilder zu schätzen weiß. Hat Bit 0 den Wert 1, kann ohne Beeinflussung der Anzeige auf den Pufferspeicher zugegriffen werden. Sitzt Bit 3 (vertikale Synchronisation) auf 1, dann befindet sich der Elektronenstrahl im vertikalen Rücklauf und es kann in den Bildschirmspeicher geschrieben werden. Dieses Register besitzt auch zwei Statussignale für Lichtgriffel.

Bits 7	6	5	4	3	2	1	0	Verwendung
.	.	.	.	.	.	.	X	1 = Speicherzugriff möglich ohne Bildschirmstörungen
.	.	.	.	.	.	X	.	1 = Lichtgriffel getriggert
.	.	.	.	.	X	.	.	0 = Lichtgriffel an; 1 = Lichtgriffel aus
.	.	.	.	X	.	.	.	1 = Elektronenstrahl ist im vertikalen Rücklauf
X	X	X	X	.	.	.	.	Unbenutzt

Tabelle 4-17 Die Kodierung des Statusregisters

Die Lichtgriffel-Latch-Ports (hex 3DB und 3DC): Der schreibende Zugriff auf einen dieser Ports löscht oder setzt eine Festumschaltung, die mit dem Lichtgriffeleingang des 6845 verbunden ist.

Der 6845 Bildschirm-Controller (hex 3D0 und 3D1): Der Controller funktioniert mit dem Farb-/Grafikadapter in derselben Weise wie mit dem Monochromadapter.

Die Steuerung des Bildschirms ist eine komplizierte Sache, und sie wird immer komplexer durch die ständigen Erweiterungen. Was Sie auch vorhaben mögen, testen Sie zunächst, ob Sie die Materie beherrschen, bevor Sie etwas in Ihr Programm endgültig aufnehmen.

4.6 Kompatibilitätserwägungen

Um die Kompatibilität von Programmen zu allen PC-Modellen von IBM sicherzustellen, bedenken Sie bitte, daß ein Standard-PC, der nur mit dem Monochromadapter ausgerüstet ist, keine Grafikdarstellungen erzeugen kann. Die Modi 13 bis 16 sind dem HR-Farbgrafikadapter (*Enhanced Graphics Adapter* oder EGA) vorbehalten! Auch das Austauschen von Farbpaletten ist nur mit dem HR-Farbgrafikadapter (EGA) möglich. Und vergessen Sie auch nicht, daß der Monochromadapter des PC und des XT die Farbattribute im Textmodus anders behandelt. Sie finden nähere Erläuterungen hierzu in Kapitel 4.3.2.1.
Es ist eine gute Idee, Programme so zu gestalten, daß sie sich den jeweiligen Hardwareumgebungen so weit wie möglich selbständig anpassen können. Für den Gebrauch von Bildschirmfarben und Text-/Grafikmodi bedeutet das, daß der IBM-Monochrommonitor ebenso besonders zu berücksichtigen ist, wie ein anderer S/W-Bildschirm mit Mischsignaleingang. Gerade bei letzteren führen Farbinformationen häufig zu völlig unzulänglichen Darstellungen. Monochrommonitore werden häufig bei PCs verwendet, die vor allem für Textverarbeitung oder textorientierte Aufgaben eingesetzt werden. Auf der Basis dieser Erkenntnissen heraus ist es für Software grundsätzlich von großem Vorteil, sich den jeweiligen Gegebenheiten der Hardwareumgebung möglichst anzupassen.
Damit sich ein Programm adaptieren kann, muß es den Bildschirmmodus herausfinden, bevor es reagieren kann. Ein schon vorbereiteter Weg hierzu steht für Programme, die eine Assemblerschnittstelle zum BIOS verwenden, mit der BIOS-Bildschirmroutine 15 (siehe auch Kapitel 9) bereit. Für alle anderen Programme wird diese Routine zum Hindernis. Das Problem kann aber umgangen werden, indem die Speicherstelle 0000:0449 gelesen wird. Dort ist der Bildschirmmodus abgespeichert. Kapitel 3.2.2 gibt hierzu weitere Auskünfte. Mit einem BASIC-Programm können Sie diese Speicherstelle mit den folgenden Befehlen lesen:

```
DEF SEG = 0
BILDMODUS = PEEK (&H449)
```

Der Bildschirmmodus 7 zeigt die Benutzung eines Monochrommonitors an. Leider gibt es keinen Automatismus, den Unterschied zwischen dem Original-Monochrommonitor von IBM und einem ähnlichen S/W-Monitor festzustellen. Der Benutzer eines solchen Gerätes kann aber das DOS-Kommando MODE zur Farbunterdrückung verwenden und ermöglicht damit einem Programm, in den Modi 0 und 2 eine solche Differenzierung durchzuführen.
Wünschen Sie, die Kompatibilität auf allen IBM PC-Modellen und allen Bildschirmen zu erhalten, müssen Sie mehrere Kompatibilitätskriterien beachten, die nicht immer miteinander in Einklang zu bringen sind. Hierbei zeigt sich die Unbeständigkeit beim Entwurf der IBM PCs und die Unmenge der möglichen Bildschirmformate. Es gibt aber einige übergeordnete Regeln.

Grundsätzlich ist zu sagen, daß reine Textdarstellungen die Kompatibilität erhöhen. Viele der PCs sind nur mit einem Monochromadapter ausgerüstet, der keine Grafik erzeugen kann. Bei der Entscheidung Nur-Text oder Text-und-Grafik sollten Sie zwei Argumente mit einbeziehen, wovon eines für die Nur-Text-Alternative spricht, das andere dagegen. Einerseits gibt es Programme, die eindrucksvoll den Beweis liefern, daß mit den Zeichen, die Ihnen in den Textmodi zur Verfügung stehen, wirklich gute Grafiken dargestellt werden können. Zu den verfügbaren Zeichen und deren Gebrauch für Grafiken finden Sie mehr in Anhang C. Andererseits wird es für mehr und mehr Computer zum Regelfall, Grafikfähigkeiten schon als normale Hardwareausstattung mit anzubieten. In Zukunft wird die ausschließliche Textverarbeitung wohl immer weiter an Gewicht verlieren und wir können (hoffentlich bald!) beliebige Grafiken einfügen, ohne uns um Kompatibilitätsprobleme kümmern zu müssen.
Man kann also sagen: Je weniger Farbe in einem Programm genutzt wird, desto höher ist die Kompatibilitätschance. Das heißt nätürlich nicht, daß Sie Farben in Ihren Programmen vermeiden müssen, sondern vielmehr, daß Farben eher als Erweiterung, denn als essentielles Mittel behandelt werden sollten. Kommen Programme ohne Farben aus, so sind sie auch zu Computern kompatibel, die nur einfarbig darstellen.
Und schließlich sollten Programme in der Lage sein, mit 40- und mit 80-Spalten-Texten zu arbeiten, um den unterschiedlichen Bildschirmen gerecht werden zu können. Das 40-Spalten-Format erhöht die Lesbarkeit, reduziert aber die anzeigbare Informationsmenge. Ist diese nur klein, kann also ruhig grundsätzlich ein 40-Spalten-Format gewählt werden. Besser ist es aber, wenn sich Programme an das jeweile Format anpassen oder es dem Benutzer ermöglichen, entsprechende Anweisungen zu geben. Beachten Sie dabei bitte, daß ein 40-Spalten-Modus nicht auf einem Gerät mit Monochromadapter läuft.
Unter Berücksichtigung dieser Regeln und der Besonderheiten Ihrer Programme müssen Sie nun selbst entscheiden, ob Sie den Vorteil einer umfassenden Kompatibilität mit einer bequemen und schnellen Programmierung, die allerdings nur in eingeschränktem Maße kompatibel ist, eintauschen wollen. Prinzipiell gesehen entscheiden sich Programmierer viel zu oft für den einfacheren Weg und reduzieren damit auf bedenkliche Weise den Einsatz ihrer Programme auf unterschiedlichen Computern.

4.7 Allgemeines über Monitore

Welcher Bildschirm oder Monitor verwendet wird, hat einen großen Einfluß auf die Programmentwürfe. Viele Monitore können weder Farbe noch Grafiken erzeugen und einige haben eine derart schlechte Bildschirmqualität, daß man beim besten Willen nur im 40-Spalten-Format arbeiten kann.

Es gibt viele unterschiedliche Monitore, die mit den PCs verwendet werden können. Die zwei Hauptkategorien, Monochrommonitor und Farbmonitor, lassen sich in vier Grundtypen aufteilen:

Direktsignalmonitore: Diese Monitore wurden entworfen, um hochauflösende Texte und Zeichen-Grafiken anzuzeigen, Grafiken, die auf Pixel-Basis arbeiten, können hiermit nicht dargestellt werden. Diese Monitore arbeiten nur mit dem Monochromadapter zusammen. Grafiken, die auf jedem anderen Monitor angezeigt werden können, lassen sich auf einem Direktsignalmonitor nur mit Hilfe einer speziellen Schnittstelle wie etwa der Hercules-Karte dargestellen.

Mischsignal-S/W-Monitore: Diese Bildschirme sind weitverbreitet, sie sind auch die billigsten. Sie werden an den Mischsignalausgang (Composite-Ausgang) des Farb-/Grafikadapters angeschlossen und erzeugen ein sehr klares einfarbiges Bild, meist grün oder bernsteinfarben. Der tragbare Compaq und der tragbare PC von IBM sind mit einem solchen Monitor ausgerüstet. Ein Monochrom-Mischsignalmonitor kann Grafiken, aber keine Farben anzeigen. Einige dieser Bildschirme bieten die Möglichkeit, anstatt Farben Grauschattierungen (bzw. grün oder bernstein) darzustellen, meist wird aber ein unrichtiges oder nicht sichtbares Bild erzeugt, wenn Farbsignale empfangen werden. Verwechseln Sie den Mischsignalmonitor nicht mit dem Direktsignal-Monochrommonitor; ersterer verwendet den Farb-/Grafikadapter, während der letztere nur mit dem Monochromadapter zusammenarbeitet.

Mischsignal-Farbmonitore und Fernsehbildschirme: Darstellungen auf Mischsignalbildschirmen werden durch ein einziges Signal erzeugt, das durch den Mischsignalausgang (Composite-Ausgang) des Farb-/Grafikadapters geschickt wird. Diese Monitore produzieren Farben und Grafiken, haben aber bestimmte Einschränkungen: Eine 80-Spalten-Darstellung ist generell unleserlich, nur bestimmte Farbkombinationen ergeben ein gutes Bild und die Auflösung der Grafiken ist sehr niedrig, so daß diese einfach gehalten werden müssen, indem niedrigauflösende Grafikmodi verwendet werden.

Obwohl der normale Fernsehbildschirm (bunt oder schwarz/weiß) technisch gesehen ein Mischsignalmonitor ist, erzeugt er in der Regel ein schlechteres Bild als spezielle Datenmonitore. Eine Darstellung muß im 40-Spalten- oder sogar im 20-Spalten-Modus erfolgen, um die Lesbarkeit zu garantieren. Fernseher werden an den Mischsignalausgang des Farb-/Grafikadapters angeschlossen, das Mischsignal muß aber noch zusätzlich mit einem HF-Adapter umgewandelt werden.

RGB-Farbmonitore: Die RGB-Monitore werden als besonders hochwertig angesehen, sie kombinieren eine qualitativ sehr gute Textanzeige mit einer hohen Auflösung bei Grafiken und Farben. RGB steht für *Rot-Grün-Blau* und RGB-Monitore tragen ihren Namen, weil sie für jedes Farbsignal eine eigene Leitung besitzen (ein Mischsignalmonitor hat nur eine einzige Leitung für alle Farben gemeinsam). Diese Leitungen werden am RGB-Ausgang des Farb-/Grafikadapters angeschlossen. Ein RGB-Monitor sehr

guter Qualität erzeugt klare Texte, deren Lesbarkeit nur noch vom Monochrommonitor übertroffen wird. Die Bilder und deren Farbqualität sind meist besser als die aller auf dem Markt erhältlichen Bildschirme, die an den Mischsignalausgang angeschlossen werden.

4.8 Der Cursor

Der blinkende Cursor ist eine Besonderheit der Textmodi und gibt die momentan aktivierte Schreibposition auf dem Bildschirm an. Der Cursor besteht aus einer Anzahl von Rasterzeilen, die die ganze Fläche eines Zeichenfeldes (das ist der Platz, der einem Zeichen zur Verfügung steht) ausfüllen. Die Größe des Zeichenfeldes variiert mit dem Bildschirmadapter, der Monochromadapter verwendet ein 9-Pixel-auf-14-Rasterzeilen großes Format, der Farb-/Grafikadapter ein 8-Pixel-auf-8-Rasterzeilen-Format. Die höhere Anzahl von Rasterzeilen im Monochrommodus erlaubt eine bessere Auflösung und eine detailliertere Darstellung des Zeichens; sehen Sie hierzu bitte auch in Anhang C nach.
Das normale Cursorformat belegt jede Rasterzeile. Es kann aber so geändert werden, daß es jede beliebige Anzahl von Rasterzeilen innerhalb eines Feldes umfaßt. Den Anfang und das Ende des Cursor können Sie auf jede gewünschte Rasterzeile setzen und dann einen *Umschlag* von der niedrigeren zur höheren Rasterzeile ausführen lassen (der sichtbare Teil des Cursor schlägt dabei "hinter dem Bildschirm" um). Man kann auf diese Weise einen Cursor erzeugen, der nur einen Teil des Zeichenfeldes ausfüllt. Dabei ist auch ein "zweiteiliger" Cursor erzeugbar, von dem je ein Stück am oberen und ein anderes am unteren Rand des Zeichenfeldes sichtbar wird. In Kapitel 9.1.2 finden Sie die Beziehung zwischen dem Zeichen und der Anzahl der Rasterzeilen erklärt sowie weitere Erläuterungen zu Zeichenfeldern.
Da der blinkende Cursor in den Textmodi auf der Hardwareebene erzeugt wird, kann er mit Hilfe der Software nur begrenzt beeinflußt werden. Sie können aber immerhin sein Äußeres und seine Bildschirmposition auf mehrere Arten verändern. Einige ROM-BIOS-Routinen stehen bereit, um die Position des Cursor zu lesen oder zu verändern (siehe Kapitel 11). Eine andere Methode ist das direkte Lesen und Schreiben in den Bildschirmspeicher (Erläuterungen zu der Speicherstelle hex 450 finden Sie in Kapitel 3.2.2). Das Cursorformat wird ebenfalls über ROM-BIOS-Routinen festgestellt und verändert. Durch direktes Lesen des Speichers kann das Format herausgefunden werden. Die anzusprechende Speicherstelle ist hex 460 und wird in Kapitel 3.2.2 näher erklärt. Die meisten Programmiersprachen offerieren Ihnen ebenfalls Möglichkeiten zur Cursormanipulation.
Wollen Sie den blinkenden Cursor, der von der Hardware bereitgestellt wird, umgehen, können Sie das Bildschirmattribut *Inversdarstellung* ver-

wenden (hex 70), wann immer der wirkliche Cursor auftritt. Auf dem Bildschirm erscheint dann ein Cursor, der nicht blinkt. Sie können aber auch die ASCII-Zeichen CHR$(219) oder CHR$(254) heranziehen, die als Blöcke auf dem Bildschirm erscheinen.
Alles, was bisher besprochen wurde, gilt für den Textcursor. In den Grafikmodi erfolgt keine Cursoranzeige. Dennoch wird aber eine logische Cursorposition aufgezeichnet, die die aktive Bildschirmposition verrät. Wie in den Textmodi kann auch hier eine BIOS-Routine verwendet werden, um das Positions-Byte (hex 450) zu lesen, und natürlich ist auch hier wieder der direkte Weg möglich.
Um einen Cursor in den Grafikmodi zu erzeugen, simulieren viele Programme, einschließlich BASIC, einen Blockcursor (nicht blinkender Cursor), indem sie eine deutlich unterscheidbare Hintergrundfarbe oder ein ASCII-Block-Zeichen verwenden.

Kapitel 5
Grundlegendes über Disketten und Festplatten

5.1 Die physikalische Struktur 96
5.2 DOS-Disketten/Platten-Formate 97
5.2.1 Standard-DOS-Formate 98
5.2.2 Quad-Density-Formate 99
5.2.3 Festplattenformat 100
5.2.3.1 Aufteilung einer Festplatte 101
5.2.3.2 Formatierung der Festplattenaufteilung 101
5.3 Die logische Struktur 103
5.4 DOS-Disketten/Platten-Organisation 104
5.4.1 Belegung von Disketten 106
5.4.2 Belegung von Festplatten 107
5.5 Die logische Struktur im Detail 108
5.5.1 Urladereintrag 108
5.5.2 Dateiverzeichnis 109
5.5.2.1 Feld 1: Dateiname 110
5.5.2.2 Feld 2: Dateinamenerweiterung 111
5.5.2.3 Feld 3: Attribute 111
5.5.2.4 Feld 4: Reserviert 112
5.5.2.5 Feld 5: Zeit 113
5.5.2.6 Feld 6: Datum 113
5.5.2.7 Feld 7: Start-Cluster-Nummer 113
5.5.2.8 Feld 8: Dateilänge 114
5.5.2.9 Unterverzeichnis 114
5.5.3 Datenbereich 115
5.5.4 Dateizuordnungstabelle 116
5.5.5 Anmerkungen zur Dateizuordnungstabelle 120
5.6 Kopierschutz 120
5.7 Anmerkungen 121

Bei den meisten Computersystemen gibt es mehrere Möglichkeiten, Daten über längere Zeiträume zu speichern. Zur Auswahl stehen häufig Magnetbänder (Kassetten), Floppydisks (Disketten) und Festplatten. Die Speichermedien unterscheiden sich in Größe und Kapazität, die Speichermethode beruht jedoch in allen Fällen auf einem Prinzip: Es werden Informationen in magnetische Signale umgesetzt, die auf der Oberfläche des Mediums in Mustern gespeichert werden. Das Aussehen der Muster ist vom verwendeten Gerät und der dazugehörigen Software abhängig.
Als der erste PC im Jahre 1981 vorgestellt wurde, war er mit einer standardisierten 5,25 Zoll Floppydiskstation, die Disketten einseitig und softsektoriert mit doppelter Speicherdichte beschrieben, ausgestattet. Die Speicherkapazität betrug nur 160 Kbyte. Im Laufe der Zeit wurde von IBM das Speichervermögen der Laufwerke beträchtlich erhöht und heute sind nicht selten in den Computersystemen Festplatten mit Kapazitäten von 10, 20 oder 30 Megabyte eingebaut.
Das Speichergerät ist natürlich wichtig, aber wesentlich interessanter für den Programmierer ist die Art und Weise, in der die Daten abgelegt werden. In diesem Kapitel beschäftigen wir uns vor allem mit zwei Speichern: der Floppydiskstation (oder Diskettenstation) und der Festplattenstation. Einige andere Speichersysteme werden kurz angesprochen. Die Informationen gelten auch für die sog. *RAM-Disks*, die eine Simulation der Speicherung auf Disketten im Hauptspeicher bewirken.

5.1 Die physikalische Struktur

Das Diskettenlaufwerk und das Betriebssystem des Computers richten die *Kapazität* der Diskette ein, die *Struktur* einer Diskette ist davon aber weitgehend unabhängig. Die Daten werden auf der Oberfläche der Diskette in einer Anzahl von konzentrischen Kreisen, die *Spuren* (*Tracks*) genannt werden, gespeichert. Jede Spur ist in Segmente, die sog. *Sektoren* (*Sectors*) unterteilt. Die Datenmenge, die aufgenommen werden kann, hängt von der Anzahl der Spuren oder der *Dichte* (*Density*) und der Größe der Sektoren ab. Die Spurendichte kann von Laufwerk zu Laufwerk variieren. Die geläufigen Laufwerke mit *doppelter Dichte* (*Double Density*) können 40 Spuren, die neuen mit *vierfacher Dichte* (*Quad-Density*) 80 Spuren aufzeichnen.
Bei den Standard 5,25 Zoll Disketten wird die Lage jeder Spur und die Anzahl der nutzbaren Diskettenseiten von dem Laufwerk unveränderbar bestimmt. Die Plazierung, Größe und Anzahl der Sektoren dagegen unterliegen der Softwarekontrolle, aus diesen Grunde werden die PC-Disketten als *softsektoriert* bezeichnet. Die Charakteristika der Diskettensektoren, ihre Größe und ihre Anzahl pro Spur, werden bei der Formatierung festgelegt. Das Formatieren kann entweder vom Betriebssystem erledigt werden oder von der dafür vorgesehenen ROM-BIOS-Routine. Dabei ist ein

teilweiser Kopierschutz leicht durch nichtstandardisierte Formatierungen zu schaffen. Das kann mit Hilfe des ROM-BIOS bewerkstelligt werden, über das Sie weitere Informationen in Kapitel 10.1.6 finden.
Bei der vom Standard-ROM-BIOS unterstützten 5,25 Zoll Diskette sind die folgenden Sektorgrößen möglich: 128 Bytes, 256 Bytes, 512 Bytes und 1024 Bytes. Die DOS-Versionen 1.00 bis 3.1 verwenden durchgängig eine Sektorgröße von 512 Bytes und es ist zu erwarten, daß dieses Format auch zukünftig beibehalten wird. Trotzdem sollte jedes Programm so flexibel gestaltet werden, daß es an Veränderungen in der Sektorengröße, insbesondere in Richtung größerer Sektoren, angepaßt werden kann.
Disketten lassen sich ein- oder zweiseitig beschreiben. Festplattensysteme können über eine oder mehrere Platten verfügen und haben somit entsprechend mehr Seiten zum Speichern. Die 10 Mbyte Festplattenstation des XT arbeitet mit zwei Magnetplatten und beschreibt alle vier Seiten.

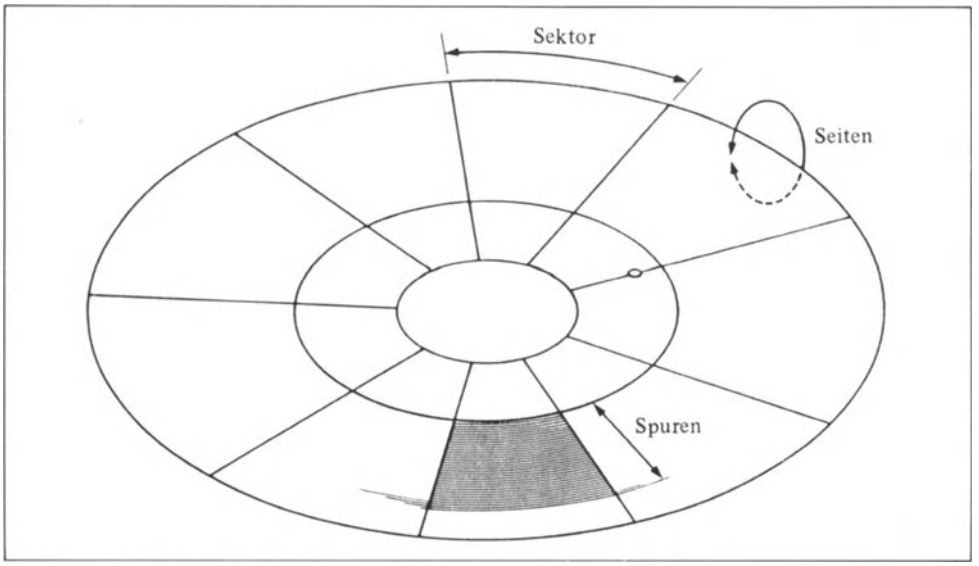


Bild 5-1 Die physikalische Struktur einer Diskette

5.2 DOS-Disketten/Platten-Formate

Die ersten von IBM vorgestellten DOS-Versionen konnten mit weniger Formaten arbeiten, als es die entsprechenden Laufwerke zugelassen hätten. DOS 2.00, das mit nur wenigen Standardformatoptionen ausgestattet ist, erlaubt es erstmals, jedes physikalische Diskettenformat zu berücksichtigen. Die gleichen Möglichkeiten stehen bei allen folgenden DOS-

Versionen zur Verfügung. Das logische Format, auf das wir noch zu sprechen kommen, ist hingegen deutlich stärker im DOS standardisiert. Die Flexibilität des Systems resultiert aus dem Konzept der sog. *installierbaren Schnittstellentreiber*. Die Treiber ermöglichen es, daß ein Laufwerk andere als standardmäßige Formate schreiben und lesen kann. Mit Schnittstellentreibern können auch andere als die IBM-Laufwerke angeschlossen werden. Weitere Informationen dazu finden Sie in Anhang B.
Die Anzahl der Diskettenformate ist mittlerweile derart groß, daß wir unmöglich alle behandeln können. Wir haben daher sieben weitverbreitete Formate herausgesucht, unter anderem ein recht spezielles, dann das geläufige 5,25 Zoll Format, ein 3,5 Zoll Minidiskettenformat und ein Festplattenformat. Diese sieben Formate sollen als Beispiele dienen und die Ausführungen müßten Sie in die Lage versetzen, mit jedem Diskettentyp zurecht zu kommen.

5.2.1 Standard-DOS-Formate

Wir beginnen mit den vier geläufigsten Formaten des PC für 5,25 Zoll Disketten. Das Format ist von der Anzahl der benutzbaren Seiten und der Sektoren abhängig. Man unterscheidet zwischen ein- und zweiseitigen Formaten und acht oder neun Sektoren pro Spur.
Sie werden sich fragen, warum gerade vier Formate? Die Antwort ist einfach: IBM mußte sicherstellen, daß alle neu erscheinenden DOS-Versionen auch die älteren PC-Modelle bedienen können. Die ersten PCs waren mit Diskettenlaufwerken ausgestattet, die nur einseitige Disketten akzeptierten. Später wurden diese Laufwerke aus dem Programm genommen und nur noch Laufwerke für Disketten ausgeliefert. Zur Erhaltung der Kompatibilität blieb das Einseitenformat aber als eines der Standardformate erhalten. Außerdem arbeitete DOS anfänglich nur mit acht 512-byte-Sektoren, obwohl gut zehn Sektoren dieser Größe auf einer Spur Platz gehabt hätten. Erst später wurden neun Sektoren zugelassen. Auch hier wurden daraufhin im Rahmen einer Gesamtkompatibilität beide Möglichkeiten (acht und neun Sektoren) offen gehalten. So kommen wir auf insgesamt vier Standardformate.
Die Formaterweiterungen resultieren also aus der Entwicklung des DOS im Laufe der Zeit. Die ursprüngliche DOS-Version 1.00 unterstützte nur das S-8-Format, die nächste Ausgabe DOS 1.10 brachte das D-8-Format hinzu. In Version 2.00 finden wir erstmals die Neun-Sektoren-Formate S 9 und D-9. Version 2.10 brachte keine Neuerungen, aber Version 3.0 arbeitet mit dem Quad-Density-Format (vierfache Speicherdichte), das wir noch näher untersuchen werden.
Die am weitesten verbreiteten Formate sind S-8 und D-9. S-8 stellt die niedrigste Spezialisierungsebene dar, d.h., das Format kann von jeder DOS-Version gelesen werden. Daher wird S-8 oftmals für kommerzielle

Bezeichnung	Seiten	Sektoren	Spuren	Speicherkapazität (Byte)
S-8	1	8	40	160 K
D-8	2	8	40	320 K
S-9	1	9	40	180 K
D-9	2	9	40	360 K

Tabelle 5-1 Die Standard-DOS-Formate

Programme verwendet, die eine weite Verbreitung finden sollen. Das D-9-Format ist wegen seiner großen Speicherkapazität das am häufigsten anzutreffende Format, besonders für Arbeitsdisketten, auf denen der Platzbedarf eine entscheidende Rolle spielt. Die beiden anderen Formate tauchen nur hin und wieder auf.

5.2.2 Quad-Density-Formate

Der konstante Faktor bei den bisher aufgezählten Formaten ist die Anzahl der Spuren, es sind 40. Die von IBM verwendeten Laufwerke sind überwiegend nur auf 40 Spuren eingerichtet. Es gibt aber heutzutage schon Laufwerke (für 5,25 Zoll und 3,5 Zoll Disketten), die mit der doppelten Anzahl von Spuren arbeiten können. Von diesen Laufwerken und den dazu gehörenden Disketten wird oft als *Quad-Density* (vierfache Datendichte) gesprochen. Unter den verschiedenen Quad-Density-Formaten sind zwei, die wir ausführlicher betrachten wollen: QD-9 und QD-15.
Das QD-9-Format ist dem D-9-Format sehr ähnlich. Die Unterschiede: QD-9 verfügt über 80 Spuren (D-9 nur 40 Spuren) auf zwei Seiten, die mit je neun Sektoren pro Spur belegt sind.
Obwohl IBM es stets vermied, daß QD-9-Format einzusetzen, ist es bei diversen Kompatiblen, wie z.B. dem DG/One-Laptop von Data General zu finden. Der DG/One verwendet 3,5 Zoll Minidisketten, die aber in ihrer logischen Struktur dem 5,25 Zoll Format entsprechen. Mit dem DG/One lassen sich daher auch die vier Standardformate verarbeiten. Die Laufwerke mit vierfacher Aufzeichnungsdichte können als nicht standardisierte Geräte mit Hilfe eines DOS-Schnittstellentreibers an den PC angeschlossen werden. Hierzu finden Sie weitere Informationen in Anhang A. Die Annahme, daß das Quad-Density-Format bald eine weite Verbreitung findet, ist nicht von der Hand zu weisen.

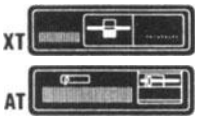
Das QD-15-Format des PC AT folgt ebenfalls weitgehend der erläuterten Struktur; es besitzt 80 Spuren pro Seite und 512-byte-Sektoren, 15 pro Spur. Die Besonderheit wird Ihnen schon aufgefallen sein, es verfügt über 15 Sektoren statt der geläufigen acht oder neun. Das wird beim AT nur durch die Verwendung spezieller 5,25 Zoll Disketten ermöglicht, die eine andersartige Magnetkodierung besitzen, die sog. *Hochkapazitätsdisketten*,

Bezeichnung	Seiten	Sektoren	Spuren	Speicherkapazität (Byte)
QD-9	2	9	80	720 K
QD-15	2	15	80	1.200 K

Tabelle 5-2 Die Quad-Density-Formate

manchmal auch *HD-Disketten* genannt (*High Density*). Nur diese Spezialdisketten, die wie gewöhnliche 5,25 Zoll Disketten aussehen, können zusammen mit geeigneten Laufwerken im QD-15-Format aufzeichnen. HD-Disketten rotieren mit einer wesentlich höheren Umdrehungsgeschwindigkeit als normale Disketten.

5.2.3 Festplattenformat



Mit dem Einsatz von Festplattenstationen hoher Kapazität wie der 10 Mbyte Festplatte des XT oder der 20 Mbyte Festplatte des AT ergeben sich spezielle Probleme, es eröffnen sich aber auch viele neue Möglichkeiten.
Zwei Aspekte sind bei allen Disketten und Platten zu berückichtigen: das physikalische und das logische Format. Das physikalische Format einer Diskette bestimmt die Sektorgröße in Byte, die Anzahl der Sektoren pro Spur (bei Festplatten: pro Zylinder), die Anzahl der Spuren (Zylinder) und die Anzahl der Seiten. Das logische Format hingegen legt fest, wie Informationen auf der Diskette organisiert und wohin verschiedene Arten von Information gespeichert werden.
Wird eine Diskette mit DOS oder einem anderen Betriebssystem formatiert, so wird das physikalische und das logische Format vorgenommen. Festplatten erhalten das physikalische Format vom Hersteller, die logische Struktur muß vom Benutzer erstellt werden. Das geschieht in zwei Schritten. Zuerst muß die Festplatte in logische Teilbereiche (sog. *Partitionen*) unterteilt werden, um die Daten und Programme jedes von uns benutzten Betriebssystems aufnehmen zu können. Wir können nämlich zusammen mit den Festplatten verschiedene Betriebssysteme benutzen, lesen Sie dazu auch Kapitel 5.2.3.1. Anschließend muß die Organisation der einzelnen Teilbereiche erfolgen, so daß jedes Betriebssystem die Informationen in seiner Partition erreichen kann. Der gesamte Organisationsvorgang wird im allgemeinen als *Formatieren* bezeichnet.

Bezeichnung	Seiten	Sektoren	Zylinder	Speicherkapazität (Byte)
XT	4	17	306	10 M
AT	4	17	615	20 M

Tabelle 5-3 Physikalische Formate der Festplatten im XT und AT

5.2.3.1 Aufteilung einer Festplatte

Jedes Betriebssystem hat eine eigene Art der Formatierung und Speicherplatzverwaltung, die nicht mit anderen Betriebssystemen kompatibel ist. Obwohl DOS das bei weitem meistgenutzte Betriebssystem des PC ist, gibt es auch andere, und es ist durchaus möglich, daß DOS eines Tages durch ein neues Betriebssystem wie z.B. XENIX abgelöst wird. Da die verschiedenen Betriebssysteme meist die gleiche Festplatte benutzen müssen, wurde ein Konzept ausgearbeitet, um eine Platte in logische Teilbereiche (Partitionen) zu splitten. Jedes Betriebssystem erhält einen Teilbereich, mit dem es arbeiten kann. Ein Teilbereich ist nichts anderes als eine Anzahl zusammenhängender Zylinder, seine Länge wird vom Benutzer bestimmt, die interne Organisation vom Betriebssystem. Nach der Unterteilung wird jeder Bereich von dem ihm zugeordneten Betriebssystem formatiert. Jedem Betriebssystem ist normalerweise ein Teilbereich zugeteilt. Leider gibt es einige Verkäufer, die den Platz auf der Festplatte in diskettenäquivalente Teilbereiche splitten und jedem das gleiche Betriebssystem, nämlich DOS, zuordnen. Das ist aber nicht der Sinn der Sache.
Das DOS-Programm FDISK erstellt die Teilbereiche und markiert einen davon für die Benutzung von DOS. Die Größe des DOS-Bereiches kann vorgegeben werden. Ist DOS das einzige Betriebssystem, so kann ihm die gesamte Plattenkapazität zugeteilt werden.
Die Anzahl und Größe der Teilbereiche kann jederzeit geändert werden, was jedoch die Zerstörung zumindest *eines* anderen Bereiches zur Folge hat. Später noch benötigte Daten sollten also vorher in einen anderen Speicher geladen werden und nach der Umverteilung der Festplatte zurückgeholt werden.

5.2.3.2 Formatierung der Festplattenaufteilung

Eine Festplatte besitzt zwei logische Strukturebenen. Die eine Ebene besteht aus dem Aufsplitten in Teilbereiche (Partitionen), die zweite Ebene ist die interne Aufteilung und Organisation der Bereiche. Während die erste Ebene allen Betriebssystemen gemeinsam ist, ist die zweite betriebssystemspezifisch. Ist ein DOS-Teilbereich auf der Platte erstellt, muß anschließendmit dem DOS-Befehl FORMAT die logische Struktur angelegt werden, die DOS zum Arbeiten benötigt.
Ähnlich wie DOS-Disketten einen Urladereintrag (Boot-Eintrag) haben, besitzen Festplatten einen Master-Urladereintrag (Master-Boot-Eintrag), der das Master-Startprogramm und die Aufteilung der Festplatte enthält. Dieser Eintrag befindet sich im ersten Sektor. Die Information über die Aufteilung besagt auch, wie viele Teilbereiche erstellt wurden (oftmals nur einer), welche Größe und Plazierung die Teilbereiche haben, von welcher Art die Teilbereiche sind und welcher von ihnen aktiv ist. Man

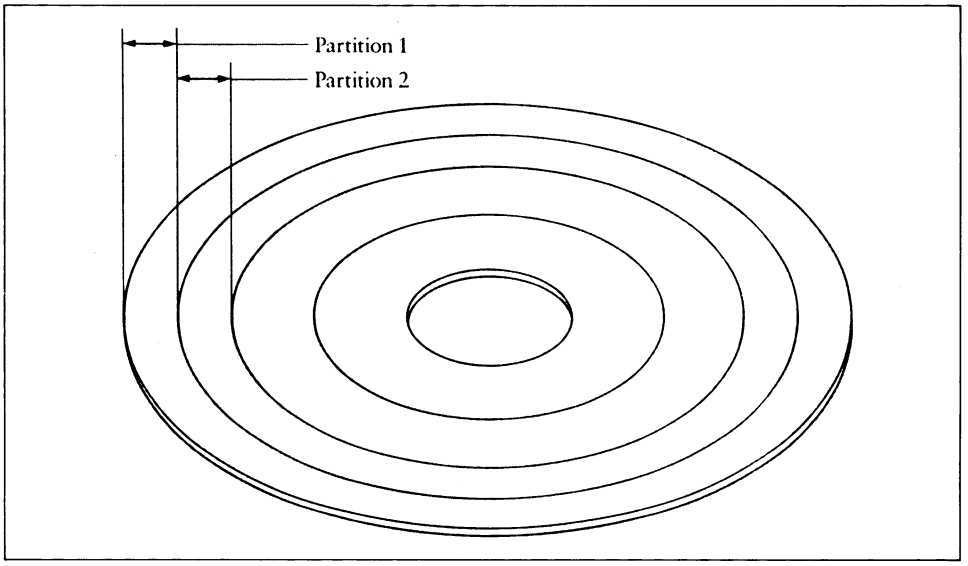
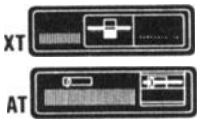


Bild 5-2 Die Einteilung einer Platte in Partitionen

findet hier auch noch andere Informationen. Das Master-Startprogramm ist ein kurzes Programm, das herausfindet, welcher Teilbereich aktiv ist und dann die Kontrolle an das Start- oder Boot-Programm dieses Teilbereiches übergibt.
So weit gelten die Anmerkungen für jede Festplatte, die in Teilbereiche unterteilt werden und mit verschiedenen Betriebssystemen arbeiten kann. Um das Prinzip der Bereichsaufteilung zu veranschaulichen, nehmen wir als Beispiel die 10 Mbyte Festplatte des XT.
Der Master-Urladereintrag dieser Festplatte enthält eine Aufteilungstabelle von bis zu vier Teilbereichen. Jeder Bereich erhält in der Tabelle eine Marke, die anzeigt, ob er gerade aktiviert ist oder nicht. Außerdem befindet sich dort ein Byte, das das in dem Bereich angelegte Betriebssystem kennzeichnet. DOS wird durch hex 01 angezeigt.
Die Position und Größe der Teilbereiche werden in der Tabelle auf zwei Arten festgehalten. Die erste Methode gibt Anfang und Ende des Teilbereiches durch Zylinder (oder Spur), Kopf (oder Seite) und Sektornummern des ersten und des letzten Sektors des Bereiches an. Die zweite Methode speichert die Sektornummer des ersten Sektors des Teilbereiches relativ zum ersten Sektor auf der Platte und die Anzahl der Sektoren dieses Bereiches.
Jeder Teilbereich behauptet für sich eine zusammenhängende Anzahl von Zylindern, er beginnt mit dem ersten Sektor des ersten Zylinders und endet beim letzten Sektor des letzten beanspruchten Zylinders. Eine Ausnahme macht der erste Teilbereich, der erst beim zweiten Sektor des ersten Zylinders anfängt, weil im ersten der Master-Urladereintrag steht.

5.3 Die logische Struktur

Unabhängig von der benutzten Diskette (bzw. Platte) ist die logische Formatierung, die DOS vornimmt, immer gleich: Die Seiten, Spuren und Sektoren werden durch Nummern festgelegt, wobei die Schreibweise stets dieselbe ist. Bestimmte Bereiche werden für Programme und Indizes reserviert, die DOS benötigt, um Diskettenoperationen ausführen zu können. Bevor wir die Diskettenorganisation näher untersuchen, wollen wir kurz besprechen, wie DOS und BIOS Informationen auf der Diskette lokalisieren können.



Die Standard 5,25 Zoll Diskette besitzt 40 Spuren, die von 0 (äußere Spur) bis 39 (innere Spur) durchnumeriert sind. Andere Diskettenformate können über mehr Spuren verfügen. Die 80 Spuren der Quad-Density-Disketten tragen Nummern von 0 bis 79, die Festplatte des XT verfügt über Zylinder von 0 bis 305. Die Zylinder der Festplatte des AT haben Nummern von 0 bis 614.

Eine zweiseitig verwendbare Diskette verfügt über die Seiten 0 und 1, einseitige Disketten haben nur die Seite 0. Die Schreib-/Leseköpfe der entsprechenden Laufwerke werden mit den gleichen Nummern versehen. Die Festplatte des XT hat vier Seiten und vier Köpfe, die die Nummern von 0 bis 3 tragen.

Die Sektoren einer Diskette sind von 1 bis 8 bzw. 9 durchnumeriert, auf der Festplatte des XT läuft die Numerierung von 1 bis 17. Beachten Sie bitte, daß die Sektornummern bei 1 beginnen, die Spur- und Seitennummern hingegen mit 0.

BIOS erkennt die Position von Sektoren durch eine dreidimensionale Koordinate. Dabei werden die Nummer der Spur, hiervon wird oft auch als *Zylindernummer* gesprochen, die Nummer der Seite, auch als *Kopfnummer* bekannt und die Sektornummer verwendet. DOS basiert auf einem anderen Konzept: Die Sektoren einer einseitigen Diskette werden ab Sektor 1 der Spur 0 auf der Seite 0 durchnumeriert; es folgen die restlichen Sektoren der gleichen Spur und Seite. Bei zweiseitigen Disketten folgt die Numerierung einem etwas anderem Schema: Dem neunten Sektor der Spur 0 auf der Seite 0 folgt der erste Sektor der Spur 0 auf der Seite 1. Auf diese Weise werden beide Seiten einer Spur durchnumeriert, bevor zur nächsten Spur übergegangen wird.

Um sich auf einen bestimmten Sektor zu beziehen, muß entweder die dreidimensionale Koordinate oder aber die laufende Numerierung angegeben werden. Alle ROM-BIOS-Befehle orientieren sich an der Koordinate, DOS und DEBUG verwenden die Durchnumerierung. In Kapitel 15.1.2 können Sie nachlesen, wie sich Angaben von der einen Notation in die andere umwandeln lassen.

Hinweis: Sie haben vielleicht bemerkt, daß wir das Wort Spur (oder Zylinder) in zwei ähnlichen Sinnzusammenhängen verwenden. Einmal meint das Wort die Datensektoren nur einer einzigen Seite auf einem konzentrisch zum Mittelpunkt verlaufenden Kreis, zum anderen meint es die Datensektoren zweier Seiten (Ober- und Unterseite) eines konzentrischen Kreises. Tatsächlich kann das Wort Spur auf beide Arten benutzt werden, Sie können meist aus dem Kontext die Bedeutung herauslesen. Auch das Wort Sektor wird im doppelten Sinne verwendet. Einmal bezeichnet es die Sektorierung über mehrere Spuren hinweg, ein andermal einen einzelnen Datensektor einer Spur. Im allgemeinen ist die zuletzt genannte Bedeutung gemeint, sofern nicht ausdrücklich etwas anderes gesagt wird.

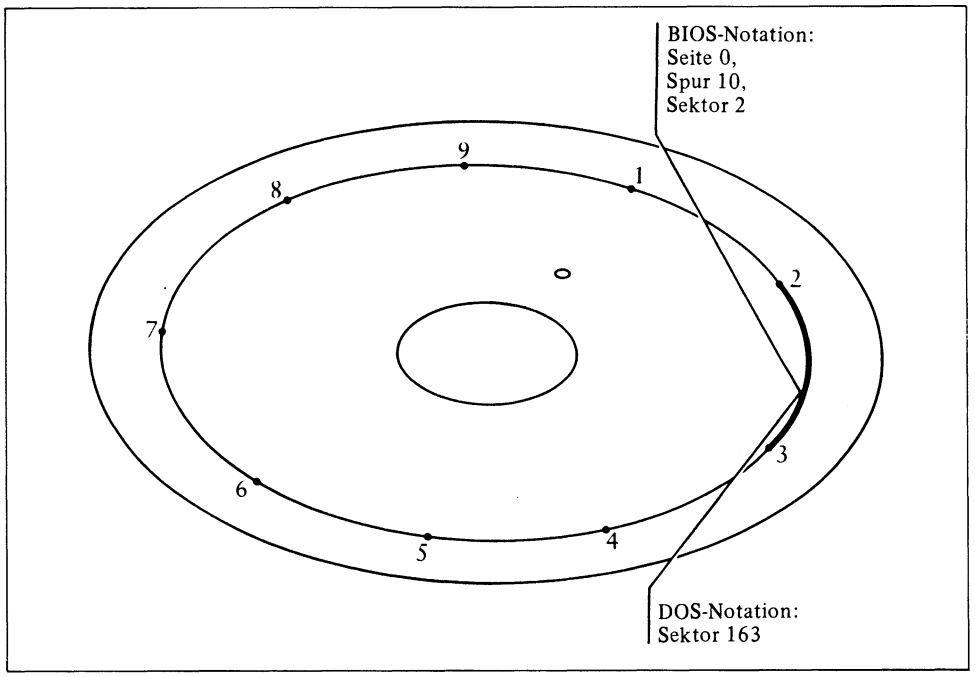


Bild 5-3 Die Sektoreinteilung gemäß ROM-BIOS und DOS

5.4 DOS-Disketten/Platten-Organisation

Bei der Diskettenformatierung unter DOS teilt das Betriebssystem jede der 40 Spuren in acht oder neun 512-byte-Sektoren ein. Die Gesamtspeicherkapazität einer D-9 Diskette beträgt rund 368.640 Bytes. Dieser Speicherbereich kann jedoch nicht vollständig zur Datenspeicherung verwendet werden. Systemkontrollinformationen und von DOS benötigte Indizes müssen dort ebenfalls abgelegt werden, damit DOS die Plazierung der und die Beziehungen zwischen den einzelnen Sektoren erkennen kann. Beim Formatieren erledigt DOS neben der Sektorunterteilung auch noch diverse andere Funktionen.

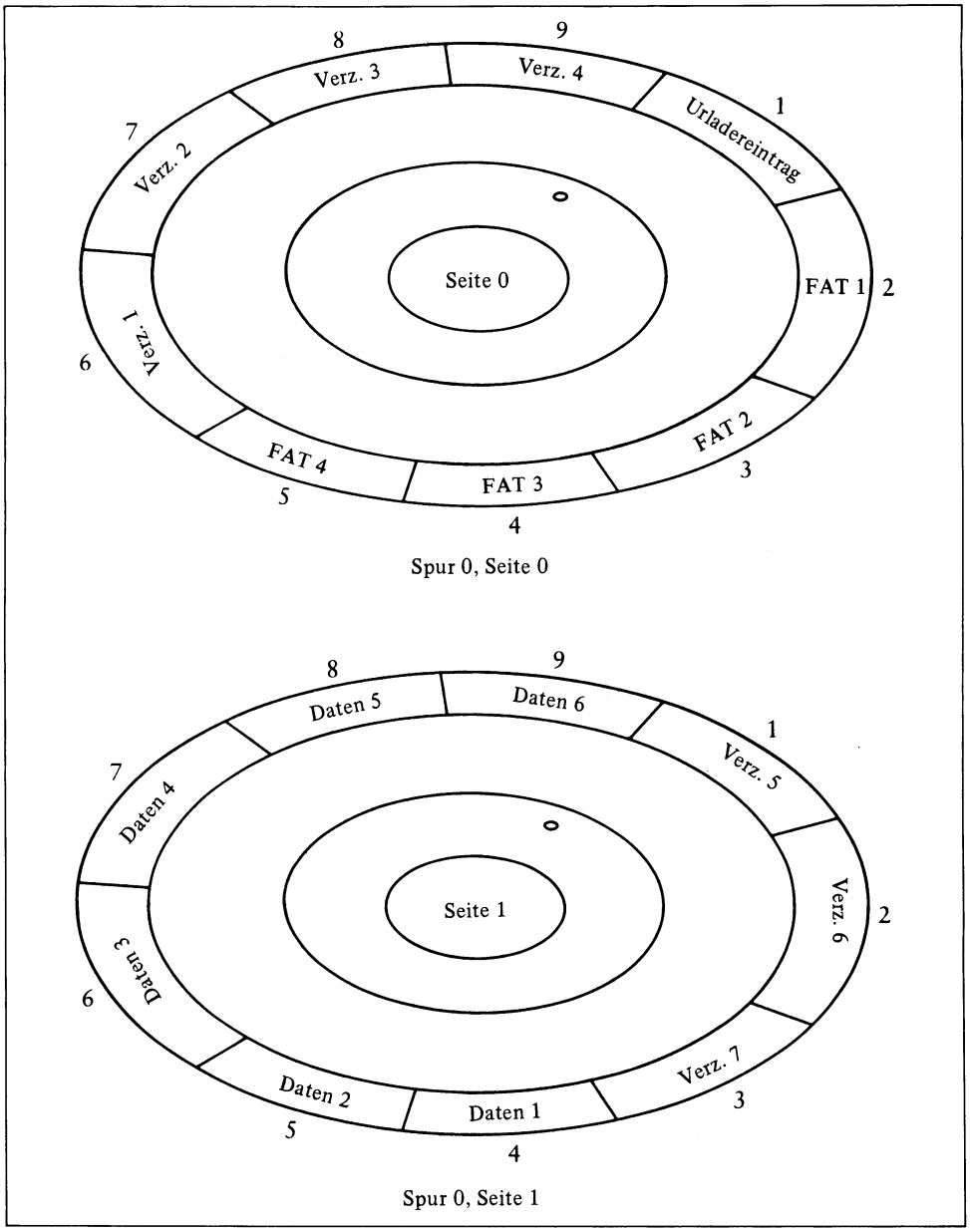


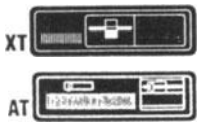
Bild 5-4 Die vier logischen Bereiche einer Diskette

5.4.1 Belegung von Disketten

Der Formatierungsprozeß unterteilt die Sektoren einer Diskette in vier Sektionen, die unterschiedlich benutzt werden. Die vier Sektionen sind der *Urladereintrag*, die *Dateizuordnungstabelle* (*File Allocation Table* oder kurz FAT), das *Dateiverzeichnis* und der *Datenspeicherbereich*. Obwohl die Länge der Sektionen innerhalb der Formate variiert, sind Struktur und Anordnung der Sektionen immer gleich. Festplatten wie z.B. die 10 Mbyte Festplatte des XT folgen ebenfalls diesem Grundschema. Hier können aber Probleme auftauchen, weil die Größe der Teilbereiche (Partitionen) direkt die Größe jeder Sektion beeinflußt. Über die Teilbereiche der Festplatten finden Sie weitere Angaben in Kapitel 5.2.3.

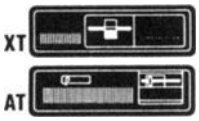
Der Urladereintrag (Boot-Eintrag) ist immer in Sektor 1 auf Spur 0 auf Seite 0 zu finden. Er beherbergt, neben anderen Dingen, ein kurzes Programm, um den Ladeprozeß des Betriebssystems von einer Betriebssystemdiskette zu beginnen. Alle Disketten besitzen den Urladereintrag, selbst wenn sie das Betriebssystem nicht enthalten. Die weiteren im Urladereintrag gegebenen Informationen sind von Format zu Format unterschiedlich.

Die Dateizuordnungstabelle oder FAT (*File Allocation Table*) folgt dem Urladereintrag und beginnt gewöhnlich in Sektor 2 auf Spur 0, Seite 0. Die FAT enthält den "offiziellen" Eintrag des Diskettenformats und ermöglicht einen Überblick über die von den Dateien belegten Sektoren. DOS benutzt die FAT, um eine Kontrolle über den von Daten belegten Speicherplatz zu erlangen. Jeder Eintrag der Tabelle enthält einen speziellen Code, der mitteilt, welcher Bereich belegt, welcher noch verfügbar ist und wo die defekten Stellen der Diskette liegen. Da mit der FAT der gesamte Datenbereich einer Diskette verwaltet wird, werden zwei völlig identische Kopien angefertigt und abgespeichert für den Fall, daß eine FAT zerstört werden sollte. Beide Kopien der FAT beanspruchen so viel Platz, wie jeweils nötig ist: zwei oder vier Sektoren auf normalen Disketten, 14 auf einer QD-15-Diskette, bis zu 16 auf der Festplatte des XT und bis zu 82 auf der Festplatte des AT. Auf allen Festplatten variiert die Größe der FAT mit der Größe der Teilbereiche (Partitionen).



Das Dateiverzeichnis dient als Inhaltsverzeichnis einer Diskette oder Platte. Jede Datei erhält einen Verzeichniseintrag, der verschiedene Informationen beinhaltet, z.B. den Namen und die Länge der Datei. Ein Teil dieses Eintrages besteht aus einer Nummer, die auf die ersten von der Datei benutzten Sektoren zeigt. Diese Nummer ist gleichzeitig auch der erste Eintrag der Datei in der Dateizuordnungstabelle FAT. Die Länge des Verzeichnisses hängt vom jeweiligen Diskettenformat ab. Eine einseitige Diskette benötigt vier Sektoren, eine zweiseitige sieben. Die Verzeichnislänge verändert sich bei Festplatten genau wie die FAT mit der Größe der Teilbereiche.

Der Datenbereich der Diskette ist zum Speichern der eigentlichen Daten bestimmt, während die zuvor behandelten Sektionen der Verwaltung dienen. Der Datenbereich umfaßt das gesamte Gebiet ab dem Dateiverzeichnis bis zum letzten Sektor der Diskette. Die Sektoren des Datenbereichs werden den Dateien je nach Notwendigkeit zugewiesen. Einer oder mehrere Sektoren zusammengefaßt (je nach Format) heißen *Cluster*. Auf einseitigen Disketten beträgt die Cluster-Länge einen Sektor, auf zweiseitigen Disketten zwei nebeneinanderliegende Sektoren. Disketten höherer Kapazität können auch mehr Sektoren zu einem Cluster zusammenfassen. Vier Sektoren bilden einen Cluster auf der 20 Mbyte Festplatte des AT, bei der 10 Mbyte Festplatte des XT können bis zu acht Sektoren zu einem Cluster gehören.



Format	Sektoren	Verwaltungssektoren				Datensektoren
		Urlader	FAT	Verzeichnis	Gesamt	
S-8	320	1	2	4	7	323
D-8	640	1	2	7	10	630
S-9	360	1	4	4	9	351
D-9	720	1	4	7	12	708
QD-9	1440	1	10	7	18	1422
QD-15	2400	1	14	14	29	2371

Tabelle 5-4 Die Verteilung der Sektoren in Standard-Diskettenformaten

5.4.2 Belegung von Festplatten

Der Bereich, den DOS mit der FAT, dem Dateiverzeichnis und dem Datenbereich belegt, variiert bei den Festplatten mit der Länge des für DOS zurückbehaltenen Teilbereiches. Der Urladereintrag hat immer eine Länge von einem Sektor.
Der einfachste Weg zum Verständnis der DOS-Aufteilung liegt in der Betrachtung unterschiedlicher Teilbereichsgrößen. Unser Beispiel bezieht sich auf die 10 Mbyte Festplatte des XT, die eine Struktur von 512 Byte pro Sektor, 17 Sektoren pro Zylinder und Seite, vier Seiten (Köpfe) pro Zylinder und 306 Zylinder pro Platte besitzt.
Tabelle 5-5 zeigt die Bereichsbelegung für drei DOS-Teilbereichsgrößen: 305 Zylinder (gesamte Festplatte des XT), 100 Zylinder und 5 Zylinder. Im allgemeinen können die Werte in der Tabelle interpoliert werden, um die Belegung für Zwischengrößen zu bekommen.

	Teilbereichslänge (Zylinder)		
	300 (komplett)	100	5
FAT-Länge (Sektoren)	8	5	1
Dateiverzeichnis (Sektoren)	32	16	4
Anzahl der Verzeichniseinträge	512	256	64
Cluster-Länge (Sektoren)	8	4	1
Anzahl der Cluster	2587	1699	333
Datenbereichslänge (Kbyte)	10348	3372	166,5

Tabelle 5-5 Bereichsbelegung für drei Teilbereiche (Partitionen) verschiedener Größe der 10 Mbyte Festplatte des XT

5.5 Die logische Struktur im Detail

Wir wollen uns nun eingehend mit den vier Sektionen einer Diskette, dem Urladereintrag, dem Dateiverzeichnis, dem Datenbereich und der Dateizuordnungstabelle beschäftigen.

5.5.1 Urladereintrag

Der Urladereintrag besteht hauptsächlich aus einem kurzen Maschinenprogramm, das das DOS-Betriebssystem in den Speicher laden soll. Um diese Aufgabe erfüllen zu können, überprüft das Urladerprogramm zuerst, ob auf der Diskette/Platte die Programme IBMBIO.COM und IBMDOS.COM vorhanden sind. Ist das der Fall, kann DOS in den Speicher transferiert und gestartet werden.
Sie können sich das Urladerprogramm mit dem DOS-Programm DEBUG ansehen. DEBUG ermöglicht es, einen beliebigen Sektor der Diskette/Platte in Assemblernotation zu lesen. Wenn Sie mehr über das Urladerprogramm erfahren wollen und mit den Kurzbefehlen von DEBUG vertraut sind, geben Sie folgende Befehle ein:

```
DEBUG
L 0 0 0 1  ; ersten Sektor laden
U 0 L 2    ; die ersten zwei Bytes disassemblieren
U 2E       ; alle Bytes ab 2E disassemblieren
```

Es werden die ersten Befehle des Urladerprogrammes auf der Diskette in Laufwerk A aufgelistet.

Hinweis: Nur beim Standard-PC beginnt das Urladerprogramm an der Stelle 2E. Für alle Modelle gilt: Die Anfangsposition des Urladerprogrammes ist aus dem zweiten Byte des ersten Sektors ersichtlich. Benutzen Sie also ein anderes Gerät als den Standard-PC, geben Sie die dritte Zeile bitte mit dem in Byte 2 des Sektors 1 gespeicherten Wertes statt 2E ein.

In den Urladereinträgen aller Formate außer S-8 und D-8 befinden sich noch einige Schlüsselparameter, die ab dem vierten Byte gespeichert sind. Diese Parameter sind Teil des BIOS-Parameterblocks, der von DOS generell benutzt wird, um Floppy-ähnliche Geräte zu bedienen. Der Rest des Urladerprogrammes ist in den ersten drei Bytes (Byte 0, 1 oder 2) und den auf den BIOS-Parameterblock folgenden Bytes untergebracht. Am Ende des Urladereintrags finden Sie ab DOS 2.0 das 2-byte-Kennzeichen hex 55 AA.

Offset	Länge	Beschreibung
3	8 Bytes	Systembezeichnung (z.B. IBM 2.1)
11	1 Wort	Anzahl der Bytes pro Sektor (z.B. 512, hex 0200)
13	1 Byte	Anzahl der Sektoren pro Cluster (z.B. 01 oder 02)
14	1 Wort	Anzahl der reservierten Sektoren am Beginn, 1 für Disketten
16	1 Byte	Anzahl der FAT-Kopien; 2 für Disketten
17	1 Wort	Anzahl der Stammverzeichniseinträge (z.B. 64 oder 112)
19	1 Wort	Gesamtzahl der Sektoren der Diskette (z.B. 720 für D-9)
21	1 Byte	Formatkennzeichen (z.B. FF, FE oder FC)
22	1 Wort	Anzahl der Sektoren pro FAT (z.B. 1 oder 2)
24	1 Wort	Anzahl der Sektoren pro Spur (z.B. 8 oder 9)
26	1 Wort	Anzahl der Seiten (Köpfe) (z.B. 1 oder 2)
28	1 Wort	Anzahl spezieller, reservierter Sektoren

Tabelle 5-6 Die Diskettenparameter im Urladereintrag

5.5.2 Dateiverzeichnis

Das Dateiverzeichnis (*Directory*) enthält die wichtigsten Informationen über die auf der Diskette/Platte gespeicherten Dateien: den Dateinamen, ihre Länge, den ersten FAT-Eintrag, Zeit und Datum der Erstellung und einige zusätzliche Daten. Die einzige dateirelevante Information, die man hier nicht findet, ist die exakte Lage der zur Datei gehörenden Cluster; diese Information ist aus der Dateizuordnungstabelle ersichtlich.
Für jede Datei existiert ein Verzeichniseintrag, der auch die Unterverzeichniseinträge beinhaltet. Jeder Eintrag besitzt eine Länge von 32 Bytes, somit kann ein Sektor 16 Einträge aufnehmen. Einseitige Disketten verfügen über Platz für 64 Einträge, da das Verzeichnis eine Gesamtlänge von vier Sektoren aufweist. Zweiseitige Disketten mit sieben Verzeichnissektoren können bis zu 112 Einträge aufnehmen. Unterverzeichnisse werden wie Dateien behandelt, es gibt daher für sie keine Beschränkung in der Anzahl der Einträge. Mehr über Unterverzeichnisse finden Sie in Kapitel 5.5.2.9. Alle 32 Byte-Einträge des Verzeichnisses werden in acht Felder unterteilt.

Offset	Feld	Beschreibung	Länge (Bytes)	Format
0	1	Dateiname	8	ASCII-Zeichen
8	2	Dateinamenerweiterung	3	ASCII-Zeichen
11	3	Attribute	1	Bit, kodiert
12	4	Reserviert	10	nicht benutzt, Nullen
22	5	Zeit	2	Wort, kodiert
24	6	Datum	2	Wort, kodiert
26	7	erster FAT-Eintrag	2	Wort
28	8	Länge der Datei	4	Ganzzahl

Tabelle 5-7 Die acht Felder eines Verzeichniseintrags

5.5.2.1 Feld 1: Dateiname

Die ersten acht Bytes des Verzeichnisses enthalten den Namen der Datei im ASCII-Format. Ist der Name kürzer als acht Zeichen, wird der fehlende Teil nach rechts hin mit Leerstellen (CHR$(32)) aufgefüllt. Es sollten stets Großbuchstaben eingegeben werden, da Kleinbuchstaben nicht immer richtig erkannt werden. Leerzeichen sollten im Dateinamen nicht auftreten. Die meisten DOS-Kommandos wie DEL oder COPY akzeptieren keine Dateinamen mit Leerzeichen zwischendrin. Trotzdem kann BASIC und auch ein Teil der DOS-Routinen mit solchen Namen arbeiten. Lesen Sie hierzu auch die Kapitel 16 und 17. Die Eigenschaft des Befehls DEL, nicht auf solche Dateinamen zu reagieren, kann ausgenutzt werden: Indem man Dateinamen mit Leerzeichen in der Mitte wählt, lassen sich Dateien erstellen, die nur mit relativ großem Aufwand wieder gelöscht werden können.
Drei Codes im ersten Byte des Dateinamens werden benutzt, um spezielle Situationen anzudeuten. Verzeichniseinträge, die noch nie belegt wurden, weisen hier den Code hex 00 auf. Dadurch kann DOS erkennen, daß keine weiteren aktiven Einträge mehr vorhanden sind, ohne das Verzeichnis bis zum Ende durchsuchen zu müssen. Zumindest gilt das ab der DOS-Version 2.0.
Ist das erste Byte eines Eintrages hex E5, zeigt das normalerweise an, daß die betreffende Datei gelöscht wurde. Da die DOS-Versionen vor 2.0 den Code hex 00 (*noch nie benutzt*) nicht verwenden, kann aber hex E5 sowohl eine gelöschte Datei als auch einen noch nicht benutzten Eintrag kennzeichnen.
Wird eine Datei gelöscht, so geschehen zwei Dinge; Das erste Byte des Dateinamens wird auf hex E5 gesetzt und der Eintrag in der Dateizuordnungstabelle wird gelöscht. Diese Tabelle ist ausführlich in Kapitel 5.5.4 erklärt. Alle anderen Informationen im Verzeichnis werden durch den Löschvorgang nicht verändert, sogar die Startnummer des ersten benutzten Clusters bleibt erhalten. Die im ersten Byte verlorengegangenen Informationen können mit Hilfe komplizierter Methoden zurückgewonnen wer-

den, solange der Eintrag nicht von einer anderen Datei benutzt wurde. Um eine neue Datei anzulegen nimmt DOS den ersten freien Eintrag, den es findet. Seien Sie also vorsichtig: Ist der gelöschte Verzeichniseintrag erst einmal wieder verwendet worden, sind Ihre Daten endgültig verloren. Der dritte Code, der vorkommen kann, ist hex 2E, das Punkt-Zeichen. Hex 2E spezifiziert ein Unterverzeichnis, weitere Informationen hierüber finden Sie im Kapitel 5.5.2.9. Steht auch im zweiten Byte der Wert hex 2E, liegt der Stammverzeichniseintrag eines Unterverzeichnisses vor. In dem Fall enthält das Feld, das auf den ersten Cluster zeigt (Feld 7), die Cluster-Nummer des Stammverzeichnisses.

5.5.2.2 Feld 2: Dateinamenerweiterung

Direkt hinter dem Dateinamen wird die standardisierte Dateinamenerweiterung gespeichert. Ihre Länge beträgt drei Bytes und wie der Dateiname selbst wird sie mit Leerstellen aufgefüllt, falls nicht alle drei Zeichen angegeben werden. Während der Dateiname mindestens aus einem regulären Zeichen bestehen muß, kann die Erweiterung vollkommen leer sein (drei Leerzeichen). Ansonsten gelten für Namenserweiterungen die gleichen Regeln wie für Dateinamen.

Hinweis: Enthält das Verzeichnis einen Eintrag des Diskettennamens, werden Dateiname und Erweiterung als ein kombiniertes Feld mit elf Bytes angesehen. In dem Fall sind eingeschlossene Leerstellen erlaubt; Kleinbuchstaben sollten eigentlich nicht, können aber hier dennoch Verwendung finden.

5.5.2.3 Feld 3: Attribute

Das dritte Feld des Verzeichnisses hat eine Länge von einem Byte. Jedes Bit (Bit 0 bis Bit 7) wird benutzt, um ein Attribut zu kennzeichnen. Tabelle 5.5.2-B gibt einen Überblick.

Bit								Wert		Bedeutung
7	6	5	4	3	2	1	0	Dez	hex	
.	.	.	.	.	.	.	1	1	1	nur lesen (Schreibschutz)
.	.	.	.	.	.	1	.	2	2	unsichtbare Datei (versteckt)
.	.	.	.	.	1	.	.	4	4	Systemdatei
.	.	.	.	1	.	.	.	8	8	Datenträgername
.	.	.	1	.	.	.	.	16	10	Unterverzeichnis
.	.	1	.	.	.	.	.	32	20	Archiv
.	1	.	.	.	.	.	.	64	40	nicht benutzt
1	.	.	.	.	.	.	.	128	80	nicht benutzt

Tabelle 5-8 Die acht Attribut-Bits einer Datei

Bit 0, das höchstwertige Bit, markiert eine Datei als *nur lesbar*. Die Datei ist somit vor Überschreiben oder Löschen geschützt. Leider beachtet DOS 1 das Kennzeichen nicht, so daß letztlich kein vollständiger Schutz gewährleistet ist.
Die Bits 1 und 2 kennzeichnen Dateien als *versteckt* oder *unsichtbar* (Bit 1) bzw. als *Systemdateien* (Bit 2). Dateien, die eine oder beide Markierungen tragen, können mit normalen DOS-Kommandos wie etwa DIR nicht erreicht werden. Programme können dennoch auf derartig markierte Dateien zugreifen, indem die Attribut-Bits im Dateikontrollblock (*File Controll Block* oder FCB) gesetzt werden. Lesen Sie hierzu auch Kapitel 16. Die beiden DOS-Dateien IBMBIO.COM und IBMDOS.COM sind auch unter den versteckten und systembezogenen Namen IO.SYS und MSDOS. SYS zu finden. Das Systemattribut (Bit 2) stammt noch aus der CP/M-Zeit und hat im Grunde mit DOS nichts zu tun.
Bit 3 markiert einen Verzeichniseintrag als den *Datenträgerkennsatz* der Diskette. Ein solcher Eintrag, der lediglich einige der acht zur Verfügung stehenden Felder ausnutzt, wird nur im Stammverzeichnis richtig erkannt. Der Datenträgername wird im Dateinamen- und im Dateinamenerweiterungsfeld gespeichert, die zu diesem Zweck als eine Einheit angesehen werden. Die Felder für die Länge und die Startnummer des ersten Clusters werden nicht benutzt, die Felder für Zeit und Datum hingegen werden verwendet.
Bit 4, das *Unterverzeichnisattribut*, wird zur Erkennung von Verzeichniseinträgen benutzt, die wiederum auf Unterverzeichnisse deuten. Da Unterverzeichnisse auf der Diskette wie normale Dateien gespeichert werden, brauchen sie zu ihrer Unterstützung den Verzeichniseintrag. Bei diesen Einträgen werden alle Felder, außer dem Feld für die Dateilänge, benutzt. Die momentane Größe eines Unterverzeichnisses finden Sie, indem Sie der Belegungskette dieser Datei in der FAT folgen.
Bit 5, das *Archivattribut*, hilft bei der Anfertigung von Sicherheitskopien (*Backup-Kopien*) von Dateien, die auf Festplatten gespeichert sind. Für alle Dateien, die seit dem letzten Backup nicht verändert wurden, ist das Bit 0. Bei Diskettendateien steht das Bit im allgemeinen auf 1, ohne hier einen besonderen Nutzen zu haben.

5.5.2.4 Feld 4: Reserviert

Dieser Bereich mit einer Länge von 10 Byte ist für zukünftige Aufgaben reserviert worden, alle Bytes stehen normalerweise auf hex 00.

5.5.2.5 Feld 5: Zeit

Feld 5 enthält einen 2-byte-Wert, der den Zeitpunkt bezeichnet, an dem die Datei erstellt oder das letztemal verändert wurde. Das Feld wird oft in Verbindung mit dem Datumsfeld benutzt. Beide Felder zusammen können eine vorzeichenlose (positive) 4-byte-Ganzzahl bilden. Diese Zahl kann mit denen anderer Verzeichniseinträge auf *größer als*, *kleiner als* oder *gleich* verglichen werden. Die Zeit wird als vorzeichenlose 2-byte-Ganzzahl behandelt, sie ist nach folgender Formel aus Stunden, Minuten und Sekunden aufgebaut:

```
Zeit = Stunde x 2048 + Minute x 32 + Sekunde / 2
```

Zur Bezeichnung der Stunde sind die Zahlen von 0 bis 23 erlaubt (24-Stunden-Uhr). Da das 2-byte-Wort für die Speicherung aller Sekunden um ein Bit zu kurz ist, werden die Sekunden in Einheiten von je 2 Sekunden abgelegt. Der gespeicherte Wert liegt zwischen 0 und 29. Ein Wert von beispielsweise 5 bedeutet also 10 Sekunden. Die Zeitangabe 11:32:10 etwa wird als 23557 gespeichert.

5.5.2.6 Feld 6: Datum

Feld 6 enthält das Datum, wann die Datei erstellt oder das letztemal verändert wurde. Die Felder 5 (Zeit) und 6 (Datum) können gemeinsam benutzt werden (siehe Feld 5). Das Datum wird als positives Ganzzahlwort behandelt. In folgender Formel zur Berechnung des gespeicherten Wertes werden Jahre, Monate und Tage benutzt:

```
Datum = (Jahr - 1980) x 512 + Monate x 64 + Tage
```

Wie Sie sehen, wird die Jahreszahl komprimiert, indem man 1980 subtrahiert. Das Jahr 1984 wird somit als ein Wert von 4 gespeichert. Mit dieser Formel würde der 12. Dezember 1984 als 2828 berechnet werden:

```
(1984 - 1980) x 512 + 12 x 64 +12 = 2828
```

5.5.2.7 Feld 7: Start-Cluster-Nummer

Das siebente Feld besteht aus einem 2-byte-Wert, der die Nummer des ersten Daten-Clusters der Datei enthält. Dieses Feld ist quasi der "Eingang" in die Belegungskette dieser Datei in der FAT. Für Dateien, die keinen Datenbereich belegen und für Datenträgerkennsatzeinträge ist die Startnummer 0 (nicht der Wert hex FFF, der in der FAT verwendet wird, um das Dateiende (*End of File*) zu kennzeichnen).

5.5.2.8 Feld 8: Dateilänge

Das letzte Feld eines Dateiverzeichniseintrages enthält die Länge der Datei, die als positive 4-byte-Ganzzahl kodiert ist. Damit lassen sich Dateilängen darstellen, die die Kapazität der Disketten überschreiten. Mithin wird die Länge einer Datei auf einer Diskette im allgemeinen durch die Diskettenkapazität begrenzt, nicht durch die Verwaltungsstruktur des DOS.
Die gespeicherte Länge ist aus DOS-Sicht immer die von der Datei benötigte Länge, tatsächlich kann der gespeicherte Wert aber größer sein als der aktuell unbedingt nötige Speicherplatz. ASCII-Textdateien, die mit Textsystemen erstellt wurden, haben als tatsächliche Endmarke oftmals das Ctrl-Z-Zeichen (CHR$(26), hex 1A). Bei diesen Dateien ist es möglich, daß das Längenattribut einen größeren Wert speichert (das nächste Vielfache von 128 Bytes). Das ist eine allgemein bekannte Praxis bei Textprogrammen, die große Datenblöcke schneller lesen und schreiben können, als wenn sie byte-weise arbeiteten. Es ist wichtig, sich klar zu machen, daß DOS das Lesen einer Datei beendet, sobald die erste Endmarke erkannt wird, entweder das Ende der Dateilänge oder das Ende der Belegungskette (hex FFF). Dabei ist es gleichgültig, welche Endzeichen zuerst auftaucht.

5.5.2.9 Unterverzeichnis

Es gibt zwei Verzeichnisarten, die Stamm- oder Hauptverzeichnisse und die Unter- oder Subverzeichnisse (*Subdirectory*). In Inhalt und Gebrauch unterscheiden sie sich grundsätzlich nicht, beide speichern die Namen und sonstigen Spezifikationen von Dateien auf der Diskette. Das unterscheidende Merkmal ist, daß das Hauptverzeichnis, zumeist einfach *Verzeichnis* genannt, in Größe und Lage festgelegt ist. Ein Unterverzeichnis hingegen ist eine Hinzufügung zum normalen Verzeichnis und kann an einer beliebigen Stelle der Diskette abgelegt werden. Jede Diskette, die mit DOS 2.00 und den folgenden Versionen benutzt wird, kann Unterverzeichnisse enthalten.
Ein Unterverzeichnis wird wie eine normale Datei angelegt. Die Feldformate und Inhalte eines Unterverzeichnisses sind identisch mit denen des Hautverzeichnisses, nur ihre nahezu unbegrenzte Länge unterscheidet sie. Ein Unterverzeichnis kann so lang wie der Datenbereich einer Diskette werden. Das wird allerdings nach Möglichkeit vermieden, da andernfalls der Datenbereich für die Dateien aufgefüllt wäre und keine anderen Informationen mehr abgelegt werden könnten. Auf HD-Disketten kommen sehr lange Unterverzeichnisse aber durchaus zum Einsatz.
Unterverzeichnisse haben immer einen Bezug zu einem übergeordneten Verzeichnis (Stammverzeichnis), das entweder ein Hauptverzeichnis oder

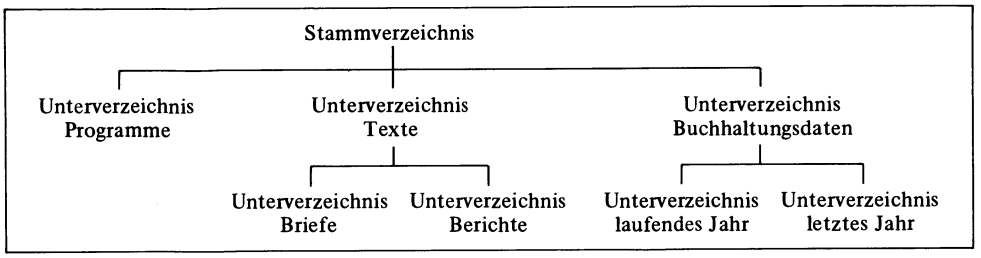


Bild 5-5 Beispiel einer Verzeichnisstruktur

ein anderes Unterverzeichnis sein kann. Man kann auf diese Weise durch Verzweigen in verschiedene Verzeichnisebenen eine baumartige Struktur aufbauen.
Stammverzeichnisse haben für jedes Unterverzeichnis einen Eintrag, der wie jeder andere Dateieintrag aussieht, außer, daß das Unterverzeichnisattribut gesetzt ist und das Dateilängenfeld auf 0 steht. Die momentane Länge des Unterverzeichnisses kann durch das Verfolgen der Belegungskette herausgefunden werden.
Ein Unterverzeichnis enthält zwei Sondereinträge mit den Dateinamen "." und "..", die bei einer Auflistung des Unterverzeichnisses mit dem DOS-Kommando DIR angezeigt werden. Der Eintrag "." kennzeichnet den Eintrag als Unterverzeichnis. Das ist notwendig, da ein Unterverzeichnis im Grunde nichts anderes als eine normale Datei ist. Der Eintrag ".." ermöglicht DOS, von dem Unterverzeichnis auf das dazugehörige Ursprungsverzeichnis zu schließen. Die Nummer des ersten Clusters verweist in beiden Einträgen auf das Unter- bzw. das Ursprungsverzeichnis. Ist die Startnummer des Clusters gleich 0, handelt es sich bei dem Ursprungsverzeichnis um ein Hauptverzeichnis.
Wird eine Datei in der Länge reduziert, kann mit DOS der unbenutzte Platz freigegeben werden. Im Falle eines Unterverzeichnisses darf unbelegter Speicherbereich nicht freigegeben werden, da andernfalls das gesamte Unterverzeichnis zerstört wird.

5.5.3 Datenbereich

Alle Dateien und Unterverzeichnisse, die sich größtenteils wie Dateien verhalten, werden im Datenbereich gespeichert.
Die Speicherkapazität wird dynamisch, das heißt, je nach Notwendigkeit, vergeben. Die kleinste Einheit dabei bildet ein Cluster. Cluster sind ein oder mehrere zusammenhängende Sektoren. Die Anzahl der Sektoren pro Cluster hängt vom jeweiligen Diskettenformat ab. Bei der Erstellung oder Vergrößerung einer Datei wächst der von dieser Datei belegte Speicherplatz. Nötigenfalls werden der Datei dabei auch neue Cluster zugewiesen.

Die DOS-Versionen 1 und 2 ordnen den jeweils ersten freien Cluster zu, spätere Versionen organisieren dies nach komplizierteren Regeln, auf die wir nicht eingehen werden.
Unter Umständen ist der von einer Datei eingenommene Bereich zusammenhängend, eine Datei kann aber auch auf mehrere getrennte Blöcke verteilt sein. Das ist vor allem dann der Fall, wenn eine schon existierende Datei erweitert oder eine neue Datei auf den Platz einer vorher gelöschten Datei geschrieben wird. Es ist also nicht ungewöhnlich, wenn die Daten einer Datei über die gesamte Diskette verteilt sind.
Die Zugriffszeit ist bei derart aufgesplitteten (man sagt auch *fragmentierten*) Dateien deutlich höher als bei anderen Dateien. Außerdem ist die Wiedergewinnung einer versehentlich gelöschten Datei durch das Suchen der einzelner Sektoren stark erschwert. Andere Nachteile treten nicht auf. Im allgemeinen interessiert es nicht, wo die einzelnen Teile der Datei auf der Diskette gespeichert sind. Wollen Sie aber doch darüber informiert sein, können Sie mit der /V-Option des DOS-Befehls CHKDSK überprüfen, ob eine Datei fragmentiert ist. Sie können aber auch Software wie z.B. die *Norton Utilities* benutzen, um eine optische Darstellung der Lage jeder Datei auf einer Diskette zu erhalten.
Haben Sie auf einer Diskette viele in Blöcke fragmentierte Dateien, können Sie sie auf eine formatierte leere Diskette kopieren, wobei gleichzeitig eine neue Anordnung geschaffen wird. Bei Disketten, die täglich benutzt und verändert werden, ist der Datenbereich natürlich in relativ kurzer Zeit aufgesplittet, so daß es ratsam ist, in regelmäßigen Zeitabständen zu kopieren. Bei Festplatten können Sie recht wenig gegen die Fragmentierung tun.
Ob Sie sich jemals mit der Aufsplittung von Dateien beschäftigen oder nicht, dieser kurze Abriß hilft Ihnen beim Verständnis der Dateizuordnungstabelle und der Belegungsketten.

5.5.4 Dateizuordnungstabelle

Die Dateizuordnungstabelle (*File Allocation Table* oder FAT), enthält Angaben zur Aufteilung der Dateien auf der Diskette. Dabei muß man zwischen der Organisation der Dateizuordnungstabelle, die relativ leicht zu verstehen ist, und ihrer Speicherung auf der Diskette, die umständlich ist, unterscheiden.
Bei den meisten Diskettenformaten werden zwei Kopien der Dateizuordnungstabelle angelegt; es können aber auch mehr sein, in Ausnahmefällen aber auch nur eine. Jede Kopie benötigt einen Sektor auf einer Acht-Sektoren-Diskette und zwei Sektoren auf einer Neun-Sektoren-Diskette. Das QD-15-Format nimmt für die Dateizuordnungstabelle und jede ihrer Kopien je sieben Sektoren in Anspruch.

Nun schreibt DOS zwar für die meisten Formate zwei Kopien der Dateizuordnungstabelle, scheint sie aber gar nicht beide zu nutzen. Das Kommando CHKDSK, das die meisten auftretenden Fehler in der Dateizuordnungstabelle und dem Verzeichnis findet, erkennt nicht einmal, wenn zwei Dateizuordnungstabellen ungleich sind.
Es gibt zwei gebräuchliche FAT-Formate: ein 12-bit-Format und ein 16-bit-Format. Die 12-bit-FAT ist geläufiger, aber auch komplexer. Die 16-bit-FAT wird immer dann verwendet, wenn die Kapazität der 12-bit-FAT überschritten wird, was z.B. bei der 20 Mbyte Festplatte des AT der Fall ist. Im folgenden werden zunächst die standardisierte 12-bit-FAT und im Anschluß die Abweichungen der 16-bit-FAT erklärt.
Die FAT ist eine Tabelle, die bis zu 4096 Nummern zwischen 0 und 4095 (hex 0 bis FFF) enthält, wobei jedem Cluster des Datenbereichs eine Nummer zugeordnet ist. Die Nummer eines Eintrags zeigt den Status und die Belegung des dem Eintrag zugeordneten Cluster an. Der Bereich der Nummern, die in der Dateizuordnungstabelle verwendet werden, überschreitet in hexadezimaler Schreibweise nicht die Grenze zur Vierstelligkeit, was ein Schlüsselelement zum Speichersystem des 12-bit-FAT-Formates ist, wie Sie gleich sehen werden.
Ein Cluster ist frei verwendbar, wenn sein FAT-Eintrag 0 ist. Ein Cluster, der wegen eines Formatierungsfehlers nicht benutzbar ist, wird mit dem Wert 4087 (hex FF7) gekennzeichnet. Die Werte von 4081 bis 4086 (hex FF1 bis FF6) sind ebenfalls reserviert, um Cluster als nicht verwendbar markieren zu können. Dem Autor ist ein solcher Fall allerdings trotz langjähriger Erfahrungen noch nicht untergekommen.
Ein Hinweis zum Thema *unbenutzbare Stellen*: Für eine Festplatte ist es normal, einige schlechte Stellen zu haben. Auf der Festplatte, auf der dieses Buch geschrieben wurde, waren drei kleine Schwachstellen auszumachen. Beim Formatierungsprozeß werden diese Stellen erkannt und entsprechend markiert. DOS erkennt die Markierung und übergeht die Schwachstellen. Bei einer Diskettenstation haben wir, anders als bei Festplatten, die Wahl, ob wir solche Disketten ausmustern und nur mit perfekten Disketten arbeiten wollen oder aber eine etwas verringerte Kapazität in Kauf nehmen.
Die Cluster sind sequentiell von zwei bis zur Gesamtzahl der auf der Diskette befindlichen Cluster plus 1 durchnumeriert. Jeder 12-bit-Eintrag einer FAT, der einen Wert zwischen 2 und 4080 (hex 02 bis FF0) enthält, entspricht einem Cluster, der einer Datei angehört. Der Wert 4095 (hex FFF) zeigt an, daß dieser Cluster der letzte einer Datei ist. Die Werte 4088 bis 4094 (hex FF8 bis FFE) sollen gleichermaßen genutzt werden können, nach der Erfahrung des Autors ist das jedoch nicht der Fall.
Mit diesem Wissen können Sie sich nun vorstellen, wie die Einträge der Dateizuordnungstabelle eine Kette formen können. Das Dateiverzeichnis enthält die Nummer des ersten Clusters einer Datei (sehen Sie hierzu auch in Kapitel 5.5.2 nach) und die Einträge der Dateizuordnungstabelle weisen

auf die weiteren von der Datei benutzten Cluster bis zum Ende der Datei. Die Cluster-Aneinanderreihung nennt man auch *Belegungskette*. Wird eine Datei gelöscht, werden alle Einträge in der Dateizuordnungstabelle als *frei verfügbar* gekennzeichnet (Wert 0). Die eigentlichen Daten im Datenbereich bleiben unverändert, ebenso die meisten Einträge im Dateiverzeichnis.
Obwohl die Dateizuordnungstabelle im Grunde eine einfache Zahlentabelle ist, wird sie in einem sehr umständlichen Verfahren abgespeichert. Dadurch erreicht man, daß die Tabelle auf der Diskette so komprimiert wie möglicht abgelegt wird und nicht viel Speicherplatz verbraucht. Hierbei kommt eine Reihe von Besonderheiten des 8088-Datenformats zur Anwendung.

Format	Sektoren	Sektoren pro Cluster	Cluster	Bereich der Cluster-Nummern
S-8	313	1	313	2 bis 314
D-8	630	2	315	2 bis 316
S-9	351	1	351	2 bis 352
D-9	708	2	354	2 bis 355
QD-9	1.422	2	711	2 bis 712
QD-15	2.371	1	2.371	2 bis 2.372

Tabelle 5-9 Die Anzahl der Cluster der verschiedenen Diskettenformate

Der Wertebereich der Dateizuordnungstabelle ist so gewählt, daß 4095 (hex FFF) nicht überschritten wird. Die dreistelligen Hex-Einträge können in 12 Bits oder 1,5 Bytes untergebracht werden. Die FAT ist paarweise organisiert, wobei jedes Eintragspaar drei Bytes beansprucht: Die Einträge 0 und 1 belegen die ersten drei Bytes, 2 und 3 die nächsten drei Bytes usw. Die Kodierung der FAT-Einträge in jeweils drei zusammengefaßten Bytes ist am einfachsten an einem Beispiel zu verdeutlichen. Angenommen, es liege das Eintragspaar hex 123 und 456 vor, so werden die

FAT-Eintrag	Wert		Bedeutung
	Dez	Hex	
0	253	FD	Doppelseitig, doppelte Datendichte
1	4.094	FFE	Eintrag unbenutzt, nicht verfügbar
2	3	3	Der nächste Cluster der Datei ist 3
3	5	5	Der nächste Cluster der Datei ist 5
4	4.087	FF7	Cluster unbenutzbar, Formatierungsfehler
5	6	6	Der nächste Cluster der Datei ist 6
6	4.095	FFF	Letzter Cluster der Datei und Ende der Belegungskette der Datei
7	0	0	Eintrag unbenutzt, verfügbar

Tabelle 5-10 Die Belegungskette für eine Beispieldatei in der Dateizuordnungstabelle

drei entsprechenden Bytes zu hex 23 61 45 . Umgekehrt: Lauten die drei Bytes AB CD EF, enhalten die zwei FAT-Einträge die Werte DAB und EFC. Bei den Formaten S-8, S-9 und D-8 ist die Anzahl der Cluster gerade, so daß ein Scheineintrag notwendig ist, damit die Paarstruktur aufgeht.

Das Schema erscheint auf den ersten Blick sehr sonderbar. Maschinensprachebefehle können damit aber sehr schnell und effizient umgehen. Ist eine Cluster-Nummer gegeben, erhalten wir den dazugehörenden Wert der FAT, indem die Cluster-Nummer mit 3 multipliziert, dann durch 2 dividiert und das Endergebnis in die FAT übernommen wird. Ist die Cluster-Nummer gerade, wird die höchstwertige Stelle abgeschnitten, ist sie ungerade, wird die niederwertige Stelle abgetrennt. Der resultierende Wert ist die nächste Cluster-Nummer der Datei, es sei denn, es wäre hex FFF, was den letzten Cluster einer Datei kennzeichnet.

Das komplexe Schema wurde eigentlich für das S-8-Format entwickelt. Für andere Formate, einschließlich der Neun-Sektoren-Formate (hier wird die FAT etwas länger als ein Sektor), ist es nicht so gut geeignet. Die bisher besprochenen Einzelheiten gelten für das 12-bit-Format, das bis zu 4080 Cluster verarbeiten kann. Für Disketten mit mehr Cluster muß das 16-bit-Format verwendet werden.

Eine 16-bit-FAT arbeitet wie die 12-bit-FAT, ist jedoch einfacher aufgebaut. Die Einträge sind hier vier Bits länger, was eine größere Anzahl von Cluster-Nummern zuläßt. 16 Bits sind exakt zwei Bytes oder ein Wort, aus diesem Grunde ist die komplizierte Speicherung der 12-bit-FAT hier nicht nötig. Die 16-bit-FAT ist eine Tabelle von Worten, die eines hinter dem anderen gespeichert sind.

Dis Sonderwerte für eine 16-bit-FAT (z.B. zur Markierung unbrauchbarer Stellen auf dem Datenträger) sind logische Erweiterungen des 12 Bit-Formats, es wird nur jeweils eine Stelle hinzugefügt (ein höchstwertiges F). Das Endkennzeichen besitzt den Wert hex FFFF statt hex FFF, wie im 12-bit-Format, die Markierung für schlechte Stellen wird zu FFF7 statt FF7.

Wie schon erwähnt werden die Cluster-Nummern von 2 ab vergeben, während die FAT ihre Numerierung mit 0 beginnt. Die ersten zwei FAT-Einträge (0 und 1) werden nicht benutzt, um den Status eines Clusters festzulegen. Vielmehr wird in Eintrag 0 ein Wert abgelegt, der das Format der betreffenden Diskette kennzeichnen soll. Leider ist die Identifizierung aber nicht eindeutig, da zum Teil unterschiedliche Formate auf dieselbe Weise gekennzeichnet werden. Daher empfiehlt es sich, das Format einer Diskette nicht auf diese Art, sondern über die DOS-Funktion 27 (hex 1B) festzustellen.

5.5.5 Anmerkungen zur Dateizuordnungstabelle

Es ist ungewöhnlich, daß Programme die FAT einer Diskette lesen oder gar verändern. Die FAT steht vollkommen unter der Kontrolle des DOS. Die einzige Ausnahme besteht in Programmen, die unabhängig von DOS Diskettenbelegungsfunktionen übernehmen, beispielsweise Programme, die gelöschte Dateien wiederherstellen wie etwa eine Routine der *Norton Utilities*.

Format	Kennzeichen-Byte
D-8	FF
S-8	FE
D-9	FD
S-9	FC
QD-9	F9
QD-15	F9

Tabelle 5-11 Die Kennzeichen-Bytes der wichtigsten Diskettenformate

Es ist wichtig zu wissen, daß die FAT beschädigt werden kann. Beispielsweise kann sich eine Belegungskette schließen, indem sie auf einen schon vorher benutzten Cluster zurückverweist oder es können zwei Ketten in einem Cluster aufeinandertreffen. Es gibt auch die Möglichkeit, daß ein Cluster *verwaist* ist, d.h., er ist als *belegt* gekennzeichnet, gehört aber keiner gültigen Kette an. Auch können Endmarken (hex FFF oder FFFF) fehlen. Die DOS-Kommandos CHKDSK und RECOVER sind zum Erkennen und Reparieren der meisten Fehler geeignet. Das gilt natürlich nur, solange die Fehler korrigierbar sind.
Die Beziehung zwischen den Belegungsketten in der FAT und Dateilängeneinträgen des DOS ist in Kapitel 5.5.2.8 erläutert.

5.6 Kopierschutz

Es gibt Dutzende von Möglichkeiten, eine Diskette davor zu schützen, daß sie kopiert wird. Die vielleicht am meisten verbreitete Methode besteht in der Neuformatierung bestimmter Spuren auf der Diskette durch die ROM-BIOS-Formatierungsroutine. Da DOS Sektoren, die mit dem eigenen Format nicht übereinstimmen, auch nicht lesen kann, können diese Disketten mit dem DOS-Kommando COPY nicht kopiert werden. Diese Beschränkung des DOS hat schon viele Firmen veranlaßt, Programme herauszubringen, die Sektoren jeder Länge lesen und kopieren können. Ein solcher Schutz ist also nicht sehr wirkungsvoll.

Auf einer höheren Ebene gibt es zwei Dinge in Bezug auf das Kopieren von Disketten, die man wissen sollte. Das erste ist, daß die meisten der exotischen und schwer zu durchbrechenden Schutzverfahren auf nicht dokumentierten Möglichkeiten des Floppydisk-Controllers (FDC) basieren. Zweitens sind manche Verfahren gewollt oder ungewollt von speziellen Eigenheiten der verschiedenen Diskettenlaufwerke abhängig. Einige Verfahren weisen eine Abhängigkeit von der Softwaresteuerung der Laufwerke auf, die innerhalb der PC-Familie keineswegs einheitlich ist. Die Software des AT unterscheidet sich beispielsweise wesentlich von der des XT und PC. Ein Kopierschutz kann also auf dem einem Modell funktionieren, während sie auf einem anderen versagt. Dieses Versagen läßt sich nur durch "Herumbasteln" am Kopierschutz und Ausprobieren an jedem Gerät vermeiden.
Wenn Sie eigene Kopierschutzverfahren entwickeln möchten: Einfallsreichtum und das Verlassen ausgetretener Wege sind der Schlüssel zum Erfolg.

5.7 Anmerkungen

Sie haben im Laufe des Kapitels eine Vielzahl technischer Details über die Struktur von Disketten und Festplatten erhalten. Die Informationen sind wichtig und Sie sollten sie sich merken (bzw. wissen, wo sie nachzuschlagen sind). Hüten Sie sich aber davor, Ihr neuerworbenes Wissen sogleich in die Tat umsetzen zu wollen. Dazu besteht nämlich im allgemeinen überhaupt kein Grund. Vielmehr sollten Sie stets auf der höchstmöglichen Ebene arbeiten, die Ihren Ansprüchen genügt. Die Reihenfolge, in der Sie dies prüfen, lautet:
Erste Wahl: Befehle, Funktionen und Bibliotheksprogramme der jeweiligen Programmiersprache (z.B. OPEN und CLOSE in BASIC);
Zweite Wahl: DOS-Routinen (Kapitel 16 und 17);
Dritte Wahl: ROM-BIOS-Routinen (Kapitel10);
Vierte Wahl: Direkte Kontrolle, z.B. das direkte Programmieren des Floppydisk-Controllers (FDC) über die Ports.
Die meisten Disketten-/Platten-Operationen können sehr bequem auf der Ebene der Programmiersprache erledigt werden. Unter zwei Bedingungen scheidet diese Möglichkeit allerdings aus. Zum einen läßt sich auf Sprachebene kein Programm schreiben, das die Laufwerkssteuerung in derselben Weise wie und parallel zum DOS durchführt. Das wäre der Fall, wenn Sie ein Programm ähnlich dem CHKDSK von DOS entwickeln wollten. Zum anderen ist es für Kopierschutzzwecke im allgemeinen unerläßlich, unkonventionelle Laufwerkssteuerungen zum Einsatz zu bringen. Hier werden häufig die ROM-BIOS-Routinen verwendet, manchmal wird aber auch der Floppydisk-Controller (FDC) direkt angesprochen (s. auch Kapitel 5.6).

Kapitel 6
Die Tastatur

6.1 Tastaturoperationen 125
6.1.1 Kommunikation mit dem ROM-BIOS 126
6.1.2 Umwandlung der Auswahlcodes 126
6.1.3 Umschaltungen 126
6.1.4 Tastenkombinationen 127
6.1.5 Tastenwiederholung 128
6.1.6 Doppelt vorhandene Tasten 128
6.1.7 Direkte Eingabe von ASCII-Zeichen 129
6.2 Tastaturdatenformat 129
6.2.1 ASCII-Tasten 129
6.2.2 Sondertasten 130
6.3 Tastaturkontrolle 132
6.3.1 Status-Bytes 132
6.3.2 Einfügemodus 133
6.3.3 Caps-Lock-Modus 133
6.3.4 Tastaturhaltestatus 133
6.3.5 Festumschaltungsstatus 134
6.4 Anmerkungen 134
6.5 Unterschiede zum AT 135

Das Kapitel behandelt hauptsächlich die Tastatur des Standard IBM PC, hin und wieder sind aber auch einige Anmerkungen zu der leicht abweichenden AT-Tastatur eingefügt. Beim AT wurden sowohl einige Tasten versetzt als auch eine neue Taste hinzugefügt; außerdem enthält die Tastatur eine etwas andere Elektronik. Die Operationseigenschaften wurden glücklicherweise nur geringfügig verändert, so daß die kleinen Eigenheiten der leicht unterschiedlichen Tastaturen für den Programmierer kaum von Bedeutung sind. Spezialmodelle wie etwa die Tastatur der 3270-PCs sind grundsätzlich ausgeklammert.

Im ersten Teil dieses Kapitels werden Sie sehen, wie die Tastatur sowohl hardware- als auch softwareseitig mit der Zentraleinheit zusammenarbeitet. Im zweiten Teil wird gezeigt, wie das ROM-BIOS mit Tastaturinformationen umgeht und und sie den Programmen zur Verfügung stellt. Falls Sie mit dem Gedanken spielen, mit direkter Tastaturkontrolle zu arbeiten, so sei Ihnen geraten, zunächst die Anmerkungen in Kapitel 6.4 zu lesen. Wie für jedes Kapitel gilt auch für das vorliegende: Setzen Sie die gegebenen Informationen nur insoweit um, wie dies sinnvoll ist. Greifen Sie also nicht zu exotischen Programmiertechniken, nur weil Sie diese einmal kennengelernt haben. Ein Beispiel für einen sinnvollen Gebrauch der direkten Tastaturkontrolle finden Sie in Programmen, die einzelne Funktionen der Tastatur ändern (Tastaturmakroprogramme). Vor einer praktischen Anwendung der im vorliegenden Kapitel gegebenen Informationen sollten Sie in jedem Fall in Kapitel 12 unter der ROM-BIOS-Tastaturroutine aufmerksam nachlesen.

Dank der offenen Softwarearchitektur des PC ist es möglich, die Tastatur sehr weitgehend zu manipulieren. Zu diesem Zweck sind eine ganze Reihe von Programmen im Handel erhältlich, wie etwa "ProKey" oder "Superkey".

Diese Programme fangen die von der Tastatur kommenden Signale ab und wandeln sie um. Es ist typisch für solche Programme, mit Tastaturmakros zu arbeiten, die angeben, auf welche Tastenanschläge geachtet werden muß und wie diese zu ändern sind. Die Umsetzung kann in einer Unterdrückung des Signals (so als wäre nie eine Taste gedrückt worden) oder dem Ersetzen eines Tastendruckes durch einen anderen (oder mehrere andere) bestehen. Der gebräuchlichste Fall ist wohl das Abkürzen von Standardformulierungen im Bereich der Textverarbeitung. So kann beispielsweise der Ausdruck "Mit freundlichen Grüßen" durch die Tastenkombination Alt-M ersetzt werden. Eine andere Anwendung liegt im Zusammenfassen mehrerer Befehlen, die dann mit einem einzigen Tastendruck aufgerufen werden können.

Die Tastaturerweiterungsprogramme vereinen in sich die Vorteile der DOS-Ebene und der ROM-BIOS-Ebene. Die DOS-Eigenschaften dieser Programme erlauben es ihnen, im Speicher zu verbleiben und von dort aus die Operationen des Prozessors zu verfolgen, während ein anderes "normales" Programm abläuft. Die ROM-BIOS-Eigenschaften ermöglichen

es, die von der Tastatur kommenden Informationen abzufangen und gegebenenfalls zu verändern, bevor sie an das andere Programm weitergegeben werden.

6.1 Tastaturoperationen

Die PC-Tastatur enthält einen eigenen Prozessor, das ist der 8048. Er ist für die Kontrolle der Tastatur verantwortlich und heißt deshalb auch *8048 Tastatur-Controller*. Die Hauptaufgabe des 8048 besteht darin, die Tastatur ständig abzufragen und im Falle einer Veränderung, dem Drükken oder Loslassen einer Taste, das ROM-BIOS zu verständigen. Wird eine Taste länger als 1/2 Sekunde niedergehalten, sendet der 8048 in periodischen Abständen ein Signal aus, wodurch die Wiederholung der Tastenfunktion veranlaßt wird. Dieser Baustein verfügt auch über eingeschränkte Diagnose- und Fehlererkennungsmöglichkeiten. Der AT benutzt einen anderen Prozessor, den 8042, der im wesentlichen aber dieselben Funktionen wie der 8048 ausübt.

AT

Jedes Drücken oder Loslassen einer Taste erzeugt im Inneren der Tastatur eine 1-byte-Zahl, die *Tastaturauswahlcode* oder *Scan-Code* genannt wird und die die Tastenbewegung eindeutig beschreibt. Die Tastatur erzeugt für das Drücken und Loslassen jeder Taste unterschiedliche Auswahlcodes. Wird eine Taste gedrückt, liegt der dieser Aktion entsprechende Code zwischen 1 und 83 (Standardtastatur). Das Loslassen erzeugt Codes, die um jeweils 128 (hex 80) höher liegen, was durch das Setzen von Bit 7 auf 1 erreicht wird. Wird der Buchstabe "Z" gedrückt, so beträgt der dazugehörende Auswahlcode 44, wird die Taste losgelassen, lautet der Code 172 (44 + 128).

Die Tastatur erkennt nicht den Sinn unseres Handelns, sie meldet nur dem ROM-BIOS, daß eine Handlung stattgefunden hat. Es ist die Aufgabe des ROM-BIOS, diese Handlungen in sinnvolle Informationen umzuwandeln, mit denen daraufhin ein Programm arbeiten kann. Die Kommunikation zwischen Tastatur und BIOS wird über Ports und Interrupts abgewickelt.

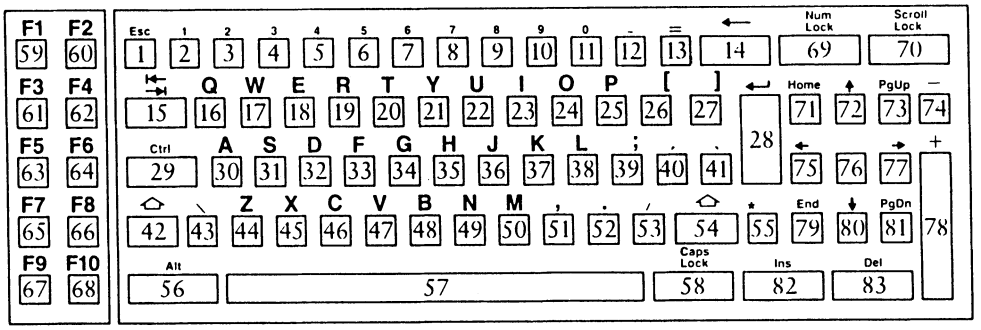


Bild 6-1 Die Standard-PC-Tastatur mit Auswahlcodes

6.1.1 Kommunikation mit dem ROM-BIOS

Wenn eine Taste auf der PC-Tastatur gedrückt oder freigegeben wird, erfährt das ROM-BIOS die Handlung über Interrupt 9, den Tastaturinterrupt. Der Interrupt 9 ruft eine Interruptunterroutine auf, die daraufhin die Nachricht, die an Port 96 (hex 60) anliegt, liest, um herauszufinden, welche Taste verändert wurde. Der Auswahlcode wird dann an das BIOS übergeben, wo er von Tastaturroutinen in einen 2-byte-Code aufgespalten wird. Dieser 2-byte-Code enthält normalerweise im niederwertigen Byte den ASCII-Wert der gedrückten Taste, im höherwertigen Byte den Auswahlcode der Tastatur. Sondertasten, wie die Funktionstasten oder die Tasten des numerischen Zehnerblocks haben eine Null im niederwertigen Byte und den Auswahlcode im höherwertigen Byte. Sie finden hierüber mehr in Kapitel 6.2.2.
Die BIOS-Routinen speichern die umgewandelten Codes in einem Pufferspeicher ab 0000:041E, der als Warteschlange fungiert. Die Codes werden hier abgelegt, bis sie von einem Programm abgerufen werden, das eine Tastatureingabe erwartet.

6.1.2 Umwandlung der Auswahlcodes

Die Kodierung der Auswahlcodes ist eine komplizierte Angelegenheit, weil es viele Umschaltungen gibt, die den Sinn eines Tastendruckes verändern können. Betätigen wir die normale Umschaltung (Shift) und "c" gleichzeitig, dann erhalten wir den Großbuchstaben "C", wird die Ctrl-Taste (*Control*) mit einem "c" gedrückt, so entspricht dies Ctrl-C oder dem *Break-Signal* (Unterbrechung). Das ist nur ein Beispiel für die verschiedenen Umschaltmodi. Man ändert während des Tippens den Umschaltmodus, indem man die Shifttaste (normale schreibmaschinenähnliche Umschaltung), die Alt-Taste oder die Ctrl-Taste drückt. Wird eine dieser Tasten betätigt und nicht wieder freigegeben, erkennt das ROM-BIOS alle folgenden Tastendrücke als mit diesem Umschaltmodus versehen.

6.1.3 Umschaltungen

Neben der "normalen" Umschaltung (Shift), der Ctrl-Taste und der Alt-Taste gibt es zwei Festumschaltungstasten, die ebenfalls den Umschaltmechanismus der Tastatur betreffen. Es sind die Tasten *Caps Lock* und *Num Lock*. Ist die Caps-Lock-Taste aktiv, kehrt sie die Bedeutung der normalen Umschalttaste für das Alphabet um, jedoch nicht für den Rest der Tastatur. Die Num-Lock-Taste schaltet die Funktion des Zehnerblocks entweder auf Cursorsteuerung oder Zahleneingabe um.

Die Statusinformation der Umschalt- und Festumschalttasten wird vom BIOS im unteren Speicherbereich bei hex 417 und hex 418 abgelegt. Wird eine dieser Tasten gedrückt, dann setzt das ROM-BIOS ein spezifisches Bit in einem dieser zwei Bytes. Sobald das ROM-BIOS den Auswahlcode für das Freigeben dieser Tasten erhält, wird das Bit wieder auf seinen Originalumschaltstatus zurückgesetzt.
Wenn das ROM-BIOS den Auswahlcode einer gewöhnlichen Tastatureingabe erhält, z.B. den Buchstaben z oder eine der Cursor-Tasten, überprüft es zuerst den Umschaltstatus und übersetzt die Eingabe dann in den entsprechenden 2-byte-Code. Diese Status-Bytes werden wir noch in Kapitel 6.3.1 ff. behandeln.

6.1.4 Tastenkombinationen

Wenn die BIOS-Routine zur Abfrage der Tastaturauswahlcodes aktiv ist, überprüft sie ständig bestimmte Umschalttastenkombinationen. Besonders wichtig sind hier die Kombinationen Ctrl-Alt-Del, Shift-PrtSc (*PrtSc* bedeutet *Drucke Bildschirm*, engl. *Print Screen*), Crtl-Num-Lock und Ctrl-Break. Diese vier befehlsähnlichen Tastatureingaben bewirken eine sofortige Unterbrechung in der Abarbeitung des ROM-BIOS, damit eine spezielle Aufgabe erledigt werden kann, die vorrangiger ist als das Puffern von Zeichen.
Crtl-Alt-Del läßt den Prozessor den Urladeprozeß nochmals durchführen, um das Kommandoprogramm erneut zu laden. Die Methode arbeitet verläßlich, solange auch die Tastatur-Interrupt-Routine arbeitet. Funktioniert die Routine nicht mehr, kann das zwei Gründe haben: Es wurde entweder der Interruptvektor (Speicherstelle hex 36 bis 39) geändert oder aber ein Befehl zum Unterdrücken von Interrupts (CLI oder *Clear Interrupt*) wurde ausgeführt, ohne daß die Interruptbearbeitung mit STI (*Start Interrupt*) wieder aktiviert wurde. In beiden Fällen haben Sie nur die Möglichkeit, durch Ausschalten des Computers und einen anschließenden Neustart von vorne zu beginnen. Das Startprogramm bereinigt alle Interrupttabellen.
Shift-PrtSc sendet den Inhalt des Bildschirms an den Standarddrucker. Diese Operation wird auf einer niedrigen BIOS-Ebene mit Interrupt 5 ausgeführt. Um die Druckerausgabe auf andere Schnittstellen umzuleiten, muß der PrtSc-Interrupt-Vektor geändert werden, so daß er auf die Routine des entsprechenden Gerätes weist. Die GRAPHICS.COM-Routine in DOS 2.00 und den folgenden Versionen umgeht den PrtSc-Befehl, indem sie zuerst den aktuellen Bildschirmmodus feststellt und im Falle eines Grafikmodus die Kontrolle einer Routine übergibt, die den Bildschirm punktweise (pixelweise) auf einen IBM-kompatiblen Grafikdrucker überträgt. Liegt kein Grafikmodus vor, wird die normale Bildschirmdruckroutine aufgerufen und die Daten werden zeichenweise übertragen.
Ctrl-Num Lock unterdrückt die Operationen des Programmes bis eine andere Taste betätigt wird.

Ctrl-Break bewirkt ein Unterbrechungssignal, indem der Interrupt 27 generiert wird. Haben Ihre Programme eine neue Interrupt-27-Routine definiert, so können sie das Signal abfangen und entsprechend handeln (oder es einfach ignorieren, je nachdem, wie Ihr Programm aussieht). Ändern Ihre Programme die Interrupt-Routine nicht, wird das laufende Programm von DOS abgebrochen.
Dies sind die einzigen wirklich bedeutungsvollen Kombinationen für das BIOS. Ungültige Kombinationen, die von der Tastatur kommen, werden einfach ignoriert und BIOS reagiert beim nächsten gültigen Tastendruck.
Nun gibt es aber noch zwei Dinge, die Sie über die Tastatur wissen müssen, bevor wir die Kodierung im Detail betrachten. Die Rede ist von den Tastenwiederholungen und den doppelt vorhandenen Tasten.

6.1.5 Tastenwiederholung

Die PC-Tastatur stellt eine automatische Tastenwiederholung bereit, die wie nachfolgend beschrieben funktioniert. Die Tastaturelektronik bemerkt, wie lange eine Taste gedrückt wird. Ist diese Zeitspanne größer als eine halbe Sekunde, wird der Auswahlcode dieser Taste zehnmal pro Sekunde wiederholt. Dieser Vorgang wird als ein aufeinanderfolgendes "Tastendrücken" gemeldet, das nicht von Tastenfreigaben unterbrochen wird. Dies macht es für eine höherentwickelte Interruptroutine möglich, einen normalen Tastendruck vom Wiederholungsprozeß zu unterscheiden. Das ROM-BIOS differenziert hier allerdings nicht immer. Die ROM-BIOS-Tastaturroutine behandelt jeden automatisch wiederkehrenden Tastendruck so, als ob die Taste tatsächlich jedesmal gedrückt worden wäre. Drücken und halten Sie z.B. die a-Taste lange genug, so daß die automatische Wiederholung beginnt, dann erkennt das ROM-BIOS eine Serie des Buchstabens a, die jedem Programm übermittelt wird, daß Tastatureingaben anfordert. Wird andererseits eine Umschalttaste gedrückt und niedergehalten, so versetzt das ROM-BIOS sich aufgrund des ersten Umschaltsignals in einen entsprechenden Umschaltmodus und ignoriert alle folgenden Signale, die von der automatischen Wiederholung stammen. An diesem Modus wird erst dann wieder etwas geändert, wenn das Freigabesignal der Umschaltung erkannt wird. Alles läuft also auf die simple Tatsache hinaus, daß das ROM-BIOS die Wiederholungssignale erkennt oder ignoriert, je nach Notwendigkeit.

6.1.6 Doppelt vorhandene Tasten

Eine Anmerkung wert ist die Tatsache, daß einige Tasten doppelt vorkommen. Zweifach vorhanden sind Sternchen (*Asterisk*), Punkt, Plus, Minus, die Ziffern von 0 bis 9 und zwei scheinbar gleiche Umschalttasten (Shifttasten).

Das ROM-BIOS wandelt die Auswahlcodes dieser doppelt vorhandenen Tasten in dieselben Zeichencodes um. Das Sternchen (*) bekommt also immer das Sternchenzeichen CHR$(42). Dennoch läßt sich feststellen, welche der beiden Tasten betätigt wurde. Die Auswahlcodes der doppelt vorhandenen Tasten, die unterschiedlich sind, werden im höherwertigen Byte abgelegt. Ein Programm braucht daher nur den Auswahlcode dieses Bytes zu testen und weiß, welche Taste gedrückt wurde. Für die beiden Shifttasten werden im Umschalt-Status-Byte (Speicherstelle hex 417) unterschiedliche Bits gesetzt. Ein Programm muß den Wert dieses Byte abfragen, um zu erfahren, welche Shifttaste betätigt wurde. Eine weitergehende Erläuterung der Speicherstelle hex 417 finden Sie in den Kapiteln 3.2.2 und 6.3.1.
Im allgemeinen kann man sich die Differenzierung zwischen zwei doppelt vorhandenen Tasten sparen. Lediglich einige sehr komplexe Programme wie Microsofts "Flight Simulator" oder Ashton-Tates "Framework" machen davon Gebrauch.

6.1.7 Direkte Eingabe von ASCII-Zeichen

Mit der PC-Tastatur können Sie ASCII-Codes direkt eingeben, also ohne den Umweg über die normalen Auswahlcodes. Zu diesem Zweck drücken Sie die Alt-Taste und tippen dann auf der rechts liegenden numerischen Tastatur die dezimalen ASCII-Zeichencodes ein. Sie können alle ASCII-Codes von CHR$(1) bis CHR$(255) eingeben. Der einzige damit nicht direkt ansprechbare ASCII-Code ist CHR$(0), da er als Signal für Nicht-ASCII-Zeichen wie Cursorsteuerung oder Funktionstasten fungiert. Dies wird im nächsten Abschnitt näher erläutert.

6.2 Tastaturdatenformat

Eine umgesetzte Tastaturbewegung (Drücken oder Loslassen) wird als Byte-Paar im Puffer des ROM-BIOS gespeichert. Zur Vereinfachung nennen wir das höherwertige Byte *Haupt-Byte*, das niederwertige Byte bezeichnen wir als *Hilfs-Byte*.

6.2.1 ASCII-Tasten

Beinhaltet das Haupt-Byte einen ASCII-Zeichenwert von CHR$(1) bis CHR$(255), so wurde eine Standardtaste gedrückt oder aber über Alt ein Zeichen des erweiterten ASCII-Codes eingegeben. Sie finden in Anhang C eine komplette ASCII-Zeichen-Tabelle. In einem solchen Falle enthält das Hilfs-Byte den Auswahlcode der gedrückten Taste, der aber normalerweise nicht benötigt wird. Die BASIC-Funktion INKEY$ fragt daher

das Hilfs-Byte gar nicht erst ab. Dennoch kann anhand des Hilfs-Bytes eine Unterscheidung der doppelt vorhandenen Tasten getroffen werden. Wurde das ASCII-Zeichen mit Hilfe der Alt-Methode eingegeben, so ist der Auswahlcode des Hilfs-Bytes gleich Null.

6.2.2 Sondertasten

Wird eine Sondertaste gedrückt, so ist das Haupt-Byte Null (CHR$(0)). Unter *Sondertasten* wollen wir folgendes verstehen: die Funktionstasten (auch umgeschaltet), die Cursor-Tasten einschließlich *Home* (bringt den Cursor an den Anfang der gerade aktuellen Zeile) und *End* (bringt den Cursor an das Ende der augenblicklichen Zeile) und viele der Ctrl- und Alt-Kombinationen. Wird eine dieser Tasten allein oder in einer Kombination gedrückt, enthält das Hilfs-Byte einen Wert, der genau diese Aktion anzeigt. Das ermöglicht Ihnen die Definition eigener Tastencodes, ohne mit den erweiterten ASCII-Zeichen (CHR$(129) bis CHR$(255)) in Konflikt zu geraten.

Hinweis: Da der Wert Null im Haupt-Byte eine Sondertaste signalisiert, kann das ASCII-Zeichen CHR$(0) auf diese Weise nicht dargestellt werden. CHR$(0) kann aber nach IBM-Definition auf der Tastatur als Alt-2 eingegeben werden. Diese Tastenkombination wird als Sondertastenwert 3 erkannt (Haupt-Byte ist 0, Hilfs-Byte ist 3). Es wird von den Programmen und Programmiersprachen erwartet, daß sie den Wert 3 des Hilfs-Bytes als CHR$(0) interpretieren. Nach der Erfahrung des Autors gibt es aber keine Programmiersprache und kein Programm, das mit dieser Konvention arbeitet.

Die Kodierung des kompletten Zeichensatz und die der Sondertasten werden vom ROM-BIOS generiert, unterschiedliche Programmiersprachen behandeln die Codes verschiedenartig. BASIC interpretiert die Sondertasten je nach Befehlsangabe unterschiedlich. Bei den regulären Eingabeanweisungen übergibt BASIC die normalen ASCII-Zeichen an das BIOS und filtert Sondertastenanschläge aus. Einige dieser Tasten können jedoch mit der Anweisung ON KEY erkannt werden. Wir können aber auch die BASIC-Funktion INKEY$ verwenden und damit direkt auf die ROM-BIOS-Kodierung der Tastaturzeichen zugreifen und herausfinden, welche Taste gedrückt wurde. Wird von der INKEY$-Funktion ein String, der nur aus einem Byte besteht, als Antwort zurückgeschickt, enthält dieser ein normales ASCII-Zeichen. Hat INKEY$ hingegen einen String aus zwei Bytes aufgenommen, so ist das erste Byte das Haupt-Byte des ROM-BIOS, welches hier immer CHR$(0) entspricht; das zweite Byte ist das Hilfs-Byte und zeigt an, welche Taste gedrückt wurde.

Hilfs-Byte-Wert (dez)	Tastenanschlag bzw. Tastaturanschläge	Hilfs-Byte-Wert (dez)	Tastenanschlag bzw. Tastaturanschläge	Hilfs-Byte-Wert (dez)	Tastenanschlag bzw. Tastaturanschläge
59	F1	110	Alt-F7	44	Alt-Z
60	F2	111	Alt-F8	45	Alt-X
61	F3	112	Alt-F9	46	Alt-C
62	F4	113	Alt-F10	47	Alt-V
63	F5	120	Alt-1	48	Alt-B
64	F6	121	Alt-2	49	Alt-N
65	F7	122	Alt-3	50	Alt-M
66	F8	123	Alt-4	3	Soll CHR$(0) sein (siehe Text)
67	F9	124	Alt-5		
68	F10	125	Alt-6	15	Shift-Tab
84	Shift-F1	126	Alt-7	71	Home
85	Shift-F2	127	Alt-8	72	Pfeil nach oben
86	Shift-F3	128	Alt-9	73	PgUp
87	Shift-F4	129	Alt-0	75	Pfeil nach links
88	Shift-F5	130	Alt-Bindestrich	77	Pfeil nach rechts
89	Shift-F6	131	Alt-=	79	End
90	Shift-F7	16	Alt-Q	80	Pfeil nach unten
91	Shift-F8	17	Alt-W	81	PgDn
92	Shift-F9	18	Alt-E	82	Ins
93	Shift-F10	19	Alt-R	83	Del
94	Ctrl-F1	20	Alt-T	114	Echo (Ctrl-PrtSc)
95	Ctrl-F2	21	Alt-Y	115	Ctrl-Pfeil nach links
96	Ctrl-F3	22	Alt-U	116	Ctrl-Pfeil nach rechts
97	Ctrl-F4	23	Alt-I	117	Ctrl-End
98	Ctrl-F5	24	Alt-O	118	Ctrl-PgDn
99	Ctrl-F6	25	Alt-P	119	Ctrl-Home
100	Ctrl-F7	30	Alt-A	132	Ctrl-PgUp
101	Ctrl-F8	31	Alt-S		
102	Ctrl-F9	32	Alt-D		
103	Ctrl-F10	33	Alt-F		
104	Alt-F1	34	Alt-G		
105	Alt-F2	35	Alt-H		
106	Alt-F3	36	Alt-J		
107	Alt-F4	37	Alt-K		
108	Alt-F5	38	Alt-L		
109	Alt-F6				

Tabelle 6-1 Die Hilfs-Byte-Werte der 97 Sondertasten der Standard-PC-Tastatur. Der Wert des Haupt-Byte beträgt immer 0.

6.3 Tastaturkontrolle

Die Tastaturoperationen, die vom ROM-BIOS überwacht werden, legen ihre Daten im unteren Speicherbereich, von hex 417 bis 472 und in den Speicherstellen hex 412 und 488, ab. Programme können diese Adressen verwenden, um den Tastaturstatus festzustellen und/oder um die Tastaturoperationen zu modifizieren. In den folgenden Abschnitten werden diese Speicherstellen auf ihre Nützlichkeit für Programme und auf die "Gefährlichkeit" von Veränderungen hin untersucht.

6.3.1 Status-Bytes

Wir beginnen mit zwei Status-Bytes für die Tastaturkontrolle in den Speicherstellen hex 417 und 418, die Sie in den Tabellen 6-2 und 6-3 finden. In diesen Status-Bytes sind die einzelnen Bits den verschiedenen Umschalt- und Festumschaltmodi zugeordnet. Alle Standardmodelle des PC verfügen über diese zwei Bytes.

Bit 7	6	5	4	3	2	1	0	Bedeutung
X	.	.	.	.	.	.	.	Einfügemodus: 1 = aktiv; 0 = inaktiv
.	X	.	.	.	.	.	.	Caps Lock: 1 = aktiv; 0 = inaktiv
.	.	X	.	.	.	.	.	Num Lock: 1 = aktiv; 0 = inaktiv
.	.	.	X	.	.	.	.	Scroll Lock: 1 = aktiv; 0 = inaktiv
.	.	.	.	X	.	.	.	Alt-Umschaltung: 1 = aktiv (Alt-Taste gedrückt); 0 = inaktiv
.	.	.	.	.	X	.	.	Ctrl-Umschaltung: 1 = aktiv (Ctrl-Taste gedrückt); 0 = inaktiv
.	.	.	.	.	.	X	.	Normalumschaltung: 1 = aktiv (linke Shifttaste gedrückt); 0 = inaktiv
.	.	.	.	.	.	.	X	Normalumschaltung: 1 = aktiv (rechte Shifttaste gedrückt); 0 = inaktiv

Tabelle 6-2 Kodierung des ersten Tastatur-Status-Bytes in der Speicherstelle hex 417

Bit 7	6	5	4	3	2	1	0	Bedeutung
X	.	.	.	.	.	.	.	1 = Ins gedrückt
.	X	.	.	.	.	.	.	1 = Caps Lock gedrückt
.	.	X	.	.	.	.	.	1 = Num Lock gedrückt
.	.	.	X	.	.	.	.	1 = Scroll Lock gedrückt
.	.	.	.	X	.	.	.	1 = Ctrl-Num Lock aktiv
.	.	.	.	.	X	.	.	1 = findet nur im PCjr Verwendung
.	.	.	.	.	.	0	.	unbenutzt
.	.	.	.	.	.	.	0	unbenutzt

Tabelle 6-3 Kodierung des zweiten Tastatur-Status-Bytes in der Speicherstelle hex 418

6.3.2 Einfügemodus

Das Bit 7 des ersten Bytes ist dem Einfügemodus vorbehalten. Es wird jedoch selten benutzt, die meisten Programme nehmen eigene Aufzeichnungen über den Einfügemodus vor.

6.3.3 Caps-Lock-Modus

Viele Programmierer erzwingen den Caps-Lock-Modus, indem sie Bit 6 im ersten Byte auf 1 setzen. Das ist für den Benutzer verwirrend, deshalb kann dies nicht unbedingt empfohlen werden. Andererseits arbeitet diese Methode zuverlässig und es gibt eine Menge guter Gründe, sich ihrer zu bedienen.

6.3.4 Tastaturhaltestatus

Der Tastaturhaltestatus ist eine interessante Eigenschaft des PC, wenngleich er kaum eine praktische Anwendung in Programmen findet. Wie bereits erwähnt, wird die Tastenkombination Ctrl-Num-Lock von BIOS besonders aufmerksam überwacht. Erscheint diese Kombination, so wird sofort in einem Modus gewechselt, der als "Tastaturhalten" bezeichnet wird. Zu diesem Zweck wird Bit 3 im ersten Status-Byte gesetzt. Das BIOS wartet nun, bis eine Taste betätigt wird, die ein darstellbares Zeichen erzeugt. Bevor dieses nicht geschehen ist, übergibt das BIOS die Kontrolle nicht an den Prozessor und es findet daher kein Programmablauf statt.
Interrupts werden in diesem Modus normal verarbeitet. Wird beispielsweise von einer Diskettenstation ein Interrupt erzeugt, der die Beendigung einer Diskettenoperation verlangt, würde die entsprechende Unterroutine den Interrupt erhalten und normal abarbeiten. Nach der Beendigung der Aufgabe geht die Kontrolle dorthin zurück, wo sie war, bevor der Interrupt ankam; das wäre die Warteschleife des BIOS auf die nächste Tastatureingabe. In diesem Haltemodus kann der Computer also auf Anforderungen externer Geräte reagieren, die laufende Programmabarbeitung ist aber im allgemeinen unterbunden. Die Tastaturroutine des BIOS wartet weiter auf Tastaturinterrupts. Sobald ein normaler Tastaturanschlag auftaucht (einschließlich der Leertaste oder einer Funktionstaste, aber keine Umschaltungen), wird der Haltemodus verlassen und das aktuelle Programm läuft weiter, als wäre nichts geschehen.
Der Haltemodus hat im Grunde keinen tatsächlichen Nutzen, vielleicht abgesehen davon, daß er dem Anwender eine einheitliche und einfache Möglichkeit in die Hand gibt, Programmpausen zu erzwingen.
Leider ist diese Methode nicht ganz sicher. Es ist nämlich für ein Programm durchaus möglich, weiterzuarbeiten, indem es einen externen In-

terrupt aufruft. Soll ein Programm unter allen Umständen ablaufen, so kann es eine Routine anwählen, die während des Haltemodus weiterarbeitet oder aber einfach den Haltemodus ausschalten.

6.3.5 Festumschaltungsstatus

Die Bits 4 bis 7 der zwei Tastatur-Bytes beziehen sich auf identische Tasten. Im ersten Byte kennzeichnen die Bits den Status der Festumschaltungen, das zweite Byte zeigt, ob die entsprechende Festumschalttaste tatsächlich in diesem Moment gedrückt ist.
Sie können die Bits lesen, aber nur wenige werden sich für Ihre Programme als nützlich erweisen. Es wäre wohl nicht sehr klug, auf die Umschaltstatus-Bits (Bits 0 bis 6 des ersten Bytes) schreibend zuzugreifen. Die Taste-gedrückt-Bits (Bits 0 bis 3 in Byte 1 und Bits 4 bis 7 in Byte 2) dürfen auf keinen Fall verändert werden, da dies zumindest potentiell äußerst zerstörerisch ist.

6.4 Anmerkungen

Möchten Sie einen besseren Einblick in die Tastaturoperationen des PC gewinnen, sollten Sie die entsprechenden ROM-BIOS-Routinen aufmerksam studieren. Nehmen Sie sich aber gleich vor einer häufigen Fehlerquelle in acht. Es gibt zwei unterschiedliche Tastaturinterrupts, der eine reagiert auf Interrupts, die von der Tastatur kommen (Interrupt 9) und legt die Tastaturdaten in den Speicher ab, der andere reagiert auf einen Interrupt, der Tastatureingaben verlangt (Interrupt 22, hex 16) und die Daten aus dem Speicher an DOS oder das ablaufende Programm übergibt. Es passiert leicht, daß diese beiden Interrupts verwechselt werden. Weiterhin gibt es viel Verwirrung mit den Interrupts 27 und 35 (hex 1B und 23), die für die Unterbrechungstastenkombination verantwortlich sind.

Interrupt Dez	Interrupt Hex	Herkunft	Verwendung
9	9	Tastatur	Signalisiert Tastaturveränderung (Drücken oder Loslassen einer Taste)
22	16	ROM-BIOS	Ruft Standard-BIOS-Tastaturroutine auf (siehe Kapitel 12)
27	1B	ROM-BIOS	Erzeugt einen Interrupt, wenn die Unterbrechungskombination unter der Kontrolle von BIOS gedrückt wurde; eine zuvor abgelegte Routine kann angesprochen werden
35	23	DOS	Ruft eine Routine auf, falls eine entsprechende angelegt und die Unterbrechungskombination unter der Kontrolle von DOS gedrückt wurde

Tabelle 6-4 Interrupts zur Tastaturkontrolle

Wie ein roter Faden zieht sich der Ratschlag durch dieses Buch, mit dem neu erworbenen Wissen nicht "wild drauflos" zu programmieren, sondern es nur dort vorsichtig einzusetzen, wo es notwendig und sinnvoll ist. Das bedeutet, daß Ihre Programme so allgemein wie nur möglich gehalten werden sollten. Gehen Sie nicht zu sehr auf die Eigenheiten des jeweiligen Computermodells ein. Benutzen Sie außerdem so weitgehend wie möglich die bereitgestellten Mittel, wie die ROM-BIOS-Routinen, zum Manipulieren der Daten, statt auf der Hardwareebene zu programmieren. Nur wenn Sie diese Ratschläge beherzigen, kommen Sie zu Programmen, die in praktisch allen Hardwareumgebungen (also auch bei unterschiedlichen Tastaturen) funktionieren.

6.5 Unterschiede zum AT

Die Tastatur des AT unterscheidet sich von der des Standard-PC nach außen hin nur wenig. Da aber die internen (Hardware-) Differenzen für die Programmierung in der Praxis kaum von Belang sind, gehen wir auf sie nicht weiter ein. So bleibt nur recht wenig zu sagen übrig.
Die AT-Tastatur unterscheidet sich nicht grundsätzlich von der Standardtastatur, einige Tasten sind versetzt worden und eine neue Taste wurde hinzugefügt. Die Neuanordnung der Tasten hat softwareseitig keinerlei Einfluß, auch nicht auf die Auswahl der für spezielle Zwecke zu benutzenden Tasten. Die für die Programmkontrolle sehr wichtige Esc-Taste wurde von der linken oberen Ecke der Tastatur in die rechte obere Ecke versetzt; für jemanden, der beide Tastaturen (die Standard-Tastatur und die des AT) verwenden muß, ist dies eine große Umstellung, aber ansonsten ohne Belang.
Die neue Taste des AT ist die Sys-Req-Taste. Sie hat keine sinnvolle Anwendung in einem Programm, sondern wurde hinzugefügt, um das Hin- und Herspringen der CPU im Multitasking-Betrieb zu aktivieren und so die speziellen Eigenheiten des Mikroprozessors 80286 zu unterstützen.
Die Verbindung zwischen dem AT und seiner Tastatur funktioniert in beide Richtungen. Die Tastatur kann Informationen zum AT schicken und der AT kann Befehle an die Tastatur senden, um damit z.B. die Tastenindikatorleuchten zu aktivieren. Es wäre unklug, an diesen Tastaturkommandos herumzuspielen.
Prognosen über zukünftige Entwicklungen zu den Geräten der IBM sind stets eine riskante Angelegenheit. So bleibt höchstwahrscheinlich das neue Layout der AT-Tastatur und dessen internationale Variationen der Standard für die Zukunft. Da, wie bereits erwähnt, die AT-Tastatur aber softwareseitig so gut wie nicht von der des Standard-PC abweicht, sind derartige Spekulationen für Programmierer im Grund nicht von Belang.

Kapitel 7
Tonerzeugung

7.1 Physikalische Betrachtung des Tones 138
7.2 Wie der Computer Töne erzeugt 139
7.2.1 Tonsteuerung des Zeitgebers 140
7.2.2 Programmierung des Zeitgebers 141
7.2.3 Aktivierung des Lautsprechers 142
7.2.4 Direkte Lautsprecherkontrolle 143
7.3 Lautstärke und Tonqualität 144

Alle Standard-PC-Modelle können mit Hilfe des programmierbaren Zeitgeber-IC (8253-5) und des eingebauten Lautsprechers einfache Töne von sich geben. Zum Verständnis der Tonerzeugung auf dem Computer sollten wir mit einer Einführung in die Grundeigenschaften des Schalls beginnen.

7.1 Physikalische Betrachtung des Tones

Töne entstehen, indem eine vibrierende Quelle Luftpartikel in Bewegung versetzt. Dehnt sich die Quelle (z.B. eine Lautsprechermembran) aus, so wird die umgebende Luft komprimiert, zieht sie sich zusammen, so reißt der Druckabfall die Teilchen auseinander. Eine Schwingung, die aus Zusammenziehen und Ausdehnen besteht, verursacht den Zusammenstoß vieler Luftpartikel. Diese Bewegung setzt sich als Kettenreaktion in alle möglichen Richtungen fort. Eine solche Fortbewegung wird *Welle* genannt, das Fortpflanzungsmedium ist die Luft.
Der Lautsprecher vibriert durch die an ihn gesendeten elektrischen Impulse des Computers. Da Computer normalerweise nur mit zwei Werten arbeiten, sind die produzierten Spannungen entweder hoch oder niedrig (Binärprinzip). Jede Spannungsänderung zieht die Lautsprechermembran zusammen oder entspannt sie. Ein Ton wird erzeugt, wenn die anliegende Spannung von niedrig zu hoch und wieder zu niedrig verändert wird. Eine einzige Vibration, bestehend aus einem An- und einem Ausimpuls, wird *Schwingung* genannt und in Hertz gemessen (ein Hertz ist eine Schwingung pro Sekunde). Durch den PC-Lautsprecher wird eine einzelne Schwingung als Klick wahrnehmbar. Ein kontinuierlicher Ton entsteht, wenn mehrere Schwingungen pro Sekunde an den Lautsprecher geschickt werden. Wächst die Anzahl der Schwingungen pro Sekunde, dann kann man die einzelnen Klicks nicht mehr unterscheiden und es entsteht ein Ton mit einer bestimmten Frequenz. Erhält der Lautsprecher beispielsweise 523 Schwingungen in der Sekunde, was einer Frequenz von 523 Hertz entspricht, hören wir einen Ton, der der Note C entspricht.
Der Durchschnittsmensch kann Töne in einer Tonhöher von ungefähr 20 bis 20.000 Hertz wahrnehmen. Der Lautsprecher des PC kann theoretisch Frequenzen von 18 bis über eine Million Hertz aussenden; der Tonbereich erstreckt sich weit über den des menschlichen Gehörs. Wie enorm diese Frequenzspanne ist, wird deutlich, wenn man sie mit der Stimme eines normalen Menschen vergleicht, die zwischen 125 und 1.000 Hertz liegt.
Der eingebaute Lautsprecher des Standard-PC bietet keine Lautstärkekontrolle und ist auch keineswegs für musikalische Darbietungen vorgesehen. Das führt dazu, daß unterschiedliche Frequenzen verschiedene Effekte haben, manche ertönen lauter als andere, wieder andere besitzen eine akurate Tonhöhe. Diese Unregelmäßigkeiten sind durch den Lautsprecher bedingt und lassen sich (per Software jedenfalls) nicht ausschalten.

Note	Frequenz	Note	Frequenz	Note	Frequenz	Note	Frequenz
C_0	16,35	C_2	65,41	C_4	261,63	C_6	1046,50
$C_{\#0}$	17,32	$C_{\#2}$	69,30	$C_{\#4}$	277,18	$C_{\#6}$	1108,73
D_0	18,35	D_2	73,42	D_4	293,66	D_6	1174,66
$D_{\#0}$	19,45	$D_{\#2}$	77,78	$D_{\#4}$	311,13	$D_{\#6}$	1244,51
E_0	20,60	E_2	82,41	E_4	329,63	E_6	1328,51
F_0	21,83	F_2	87,31	F_4	349,23	F_6	1396,91
$F_{\#0}$	23,12	$F_{\#2}$	92,50	$F_{\#4}$	369,99	$F_{\#6}$	1479,98
G_0	24,50	G_2	98,00	G_4	392,00	G_6	1567,98
$G_{\#0}$	25,96	$G_{\#2}$	103,83	$G_{\#4}$	415,30	$G_{\#6}$	1661,22
A_0	27,50	A_2	110,00	A_4	440,00	A_6	1760,00
$A_{\#0}$	29,14	$A_{\#2}$	116,54	$A_{\#4}$	466,16	$A_{\#6}$	1864,66
B_0	30,87	B_2	123,47	B_4	493,88	B_6	1975,53
C_1	32,70	C_3	130,81	C_5	523,25	C_7	2093,00
$C_{\#1}$	34,65	$C_{\#3}$	138,59	$C_{\#5}$	554,37	$C_{\#7}$	2217,46
D_1	36,71	D_3	146,83	D_5	587,33	D_7	2349,32
$D_{\#1}$	38,89	$D_{\#3}$	155,56	$D_{\#5}$	622,25	$D_{\#7}$	2489,02
E_1	41 20	E_3	164,81	E_5	659,26	E_7	2637,02
F_1	43,65	F_3	174,61	F_5	698,46	F_7	2793,83
$F_{\#1}$	46,25	$F_{\#3}$	185,00	$F_{\#5}$	739,99	$F_{\#7}$	2959,96
G_1	49,00	G_3	196,00	G_5	783,99	G_7	3135,96
$G_{\#1}$	51,91	$G_{\#3}$	207,65	$G_{\#5}$	830,61	$G_{\#7}$	3322,44
A_1	55,00	A_3	220,00	A_5	880,00	A_7	3520,00
$A_{\#1}$	58,27	$A_{\#3}$	233,08	$A_{\#5}$	932,33	$A_{\#7}$	3729,31
B_1	61,74	B_3	246,94	B_5	987,77	B_7	3951,07
						C_8	4186,01

Wohltemperierte chromatische Tonskala; A4 = 440 kHz Alle Frequenzangaben in kHz

Bild 7-1 Notentabelle über vier Oktaven mit Frequenzangaben

7.2 Wie der Computer Töne erzeugt

Dem PC lassen sich auf zwei verschiedenen Wegen Töne entlocken. Die erste Methode besteht darin, ein Programm den Lautsprecher an- und ausschalten zu lassen, indem zwei Lausprecherbits des programmierbaren Peripherie-Interface-Bausteines (PPI) manipuliert werden. Wird diese Methode angewendet, steuert das Programm die Impulse, die zum Lautsprecher gelangen und damit die resultierende Frequenz. Die andere Methode liegt in der Verwendung des eingebauten programmierbaren Zeitgeberbausteins (Timer 8253-5), um dem Lautsprecher die Impulse mit einer präzisen Frequenz zu senden. Dieser zweiten Möglichkeit ist der Vorrang zu geben, da sie zwei nicht zu übersehende Vorteile mit sich bringt. Zum einen werden die Lautsprecherimpulse vom Zeitgeber-IC und nicht von einem Programm überwacht, die CPU ist also frei für andere Zwecke. Zum anderen ist der Zeitgeber von der Arbeitsgeschwindigkeit des Prozessors, die beim AT höher als beim PC ist, unabhängig.

Beide Methoden können sowohl miteinander als auch jede für sich eingesetzt werden, um eine Vielzahl von Tönen zu erzeugen. In diesem Kapitel wollen wir uns sowohl mit der Steuerung des Zeitgebers als auch mit der direkten Lautsprecheransteuerung befassen.

7.2.1 Tonsteuerung des Zeitgebers

Der programmierbare Zeitgeber 8253-5 ist das Herzstück des Standard-PC zur Tonerzeugung, er ist aber gleichzeitig auch für die Echtzeituhr von Bedeutung. Obgleich wir uns hier auf seine Verwendung als Tongenerator konzentrieren, ist die Hauptaufgabe des 8253-5 die Zeitmessung. Die Möglichkeit zur Tonerzeugung ist daher im Grunde ein nützlicher Nebeneffekt.
Der 8253 erhält ein Signal des Hauptoszillators des Computers, des 8284A, der mit einer Frequenz von 1.193.180 mal in der Sekunde, oder 1,193 MHz (Megahertz oder Millionen Hertz) schwingt. Der Zeitgeber ist darauf programmiert, alle 65.535 Hauptschwingungen, ungefähr 18,2 mal in der Sekunde, einen Taktinterrupt (Interrupt 8) zu produzieren. Das ROM-BIOS achtet auf diese Taktimpulse und berechnet die Tageszeit, indem der Impulszähler bei jedem Takt um eins erhöht wird. Vom ROM-BIOS wird wiederum ein Interrupt erzeugt, nämlich der Taktimpulsinterrupt (Interrupt 28).
Der ROM-BIOS-Taktimpulsinterrupt wird von vielen Programmen verwendet, um über die aktuelle Zeit informiert zu sein. Manche Programme übergehen den Interrupt 28 und arbeiten direkt mit dem Zeitgeber-IC zusammen. BASIC etwa benutzt den Zeitgeber, um die Länge eines Tones, die in Taktimpulsen gemessen wird, zu zählen. Da die Standardfrequenz von 18,2 Impulsen pro Sekunde oft nicht ausreicht, um exakte Töne zu erzeugen, wird der Zeitgeber von BASIC umprogrammiert. Er ist dann viermal schneller und sendet seinen Interrupt 8 genau 72,8 mal in der Sekunde aus. Da BASIC mit der vierfachen Geschwindigkeit zählt, können Musikstücke akkurat im Tempo reproduziert werden.

Anmerkung: BASIC vervierfacht die Taktfrequenz bei der Ausführung des Befehls MUSIC. Es vermeidet den Zusammenstoß mit dem BIOS-Taktimpulsinterrupt 28, der für andere Systemfunktionen lebenswichtig ist, indem es den Vektor des Interrupt 8 verändert. Er zeigt dann auf eine Routine, die dem ROM-BIOS nur jeden vierten Takt anzeigt. Beim vierten Impuls übergibt die Interruptbearbeitung die Kontrolle an BIOS, so daß es seinen Zählerstand erhöhen und zum richtigen Zeitpunkt den Interrupt 28 durchführen kann. Die Kontrolle geht anschließend wieder an BASIC zurück.

7.2.2 Programmierung des Zeitgebers

Zwei Schritte sind erforderlich, um mit Hilfe des Zeitgeberbausteines Töne zu erzeugen. Zuerst muß der Zeitgeber auf eine Frequenz programmiert und dann die Ausgabe des Zeitgebers auf den Lautsprecher gerichtet werden. Diese zwei vorbereitenden Schritte können einzeln durchgeführt werden. Der Ton wird ausgesendet, wenn beide Voraussetzungen (Frequenzangabe vorhanden, Zeitgeberausgabe auf Lautsprecher gerichtet) erfüllt sind und beendet, sobald eine der beiden nicht mehr gegeben ist.
Der Zeitgeber kann auf das Aussenden von Impulsen beliebiger Frequenz programmiert werden. Jeder Ton erklingt solange, bis er durch die Software abgeschaltet wird.
Wir wollen den Zeitgeber zur Tonerzeugung auf dieselbe Art wie BASIC programmieren, um die Taktimpulse zu erhalten: Wir geben ihm eine Zahl (einen anzustrebenden Zählerstand). Der Zeitgeber zählt die Systemimpulse (die mit einer Frequenz von 1,193 MHz oszillieren), bis die vorgegebene Zahl erreicht ist. Dann gibt er statt des Interrupt einen Impuls aus und beginnt den Zählvorgang wieder bei 0. Dabei dividiert der Zeitgeber die Zählerstandvorgabe durch die Systemfrequenz, um seine Ausgabefrequenz zu erhalten. Das Endergebnis ist eine Impulsserie, die einen Ton bestimmter Frequenz erzeugt, wenn der Lautsprecher angeschaltet ist.
Der Kontrollzählerstand und die resultierende Frequenz stehen in reziprokem Verhältnis zueinander:

Zählerstand = 1.193.180 / Frequenz
Frequenz = 1.193.180 / Zählerstand

Aus den beiden Formeln kann man ersehen, daß ein hochfrequenter Ton mit einem niedrigen Zählerstand erreicht wird, wohingegen tiefe Töne durch hohe Zählerstände erzeugt werden. Ein Wert von 100 würde eine hohe Frequenz von ca. 11.931 Schwingungen in der Sekunde ergeben, ein Wert von 10.000 hätte eine relativ niedrige Frequenz von etwas über 119 Schwingungen pro Sekunde zur Folge.
BASIC ist in der Lage, Frequenzen von 37 bis 32.767 Hertz zu erzeugen. Das folgende Programm zeigt, daß der interne Lautsprecher einen kleineren Frequenzbereich als BASIC abdeckt.
Ist der Wert für die gewünschte Frequenz erst einmal bekannt, muß er nur noch an das 8253-Zeitgeberregister geschickt werden. Dies geschieht über drei Portausgaben. Die erste Ausgabe, die Zahl 182 (hex B6) geht an Port 67 (hex 43) und signalisiert dem Zeitgeber, daß nun der eigentliche Wert kommt. Nun folgen das nieder- und das höherwertige Byte (in dieser Reihenfolge) des von uns berechneten Wertes. Diese zwei Ausgaben, die zusammen ein positives 16-bit-Wort bilden, werden zum Port 66 (hex 42) gesendet. Das BASIC-Programm zeigt Ihnen diesen Ablauf:

```
10 ZAEHLER = 1193280! / 3000      '3000 ist die Frequenz
20 LSB.WERT = ZAEHLER MOD 256     'Wert des niederwertigen Byte errechnen
30 MSB.WERT = ZAEHLER \ 256       'Wert des höherwertigen Byte durch
                                   Integerdivision errechnen
40 OUT 67, 182                    'Zeitgeber initialisieren
50 OUT 66, LSB.WERT               'niederwertiges Byte laden
60 OUT 66, MSB.WERT               'höherwertiges Byte laden
```

7.2.3 Aktivierung des Lautsprechers

Nachdem der Zeitgeber programmiert ist, muß der Lautsprecher aktiviert werden, um das vom Zeitgeber ausgesendete Signal umsetzen zu können. Wie die meisten anderen Teile des Computers wird der Lautsprecher durch das Senden bestimmter Werte an bestimmte Ports beeinflußt. Der hier verwendete Port hat die Nummer 97 (hex 61). Die Überwachung übernimmt der programmierbare Peripherie-Interface-Baustein (PPI). Es werden zu dem von uns hier angesprochenen Zweck nur zwei der acht Port-Bits benutzt, nämlich die niederwertigen Bits 0 und 1. Die verbleibenden Bits dienen anderen Zwecken und sollten daher unangetastet bleiben.

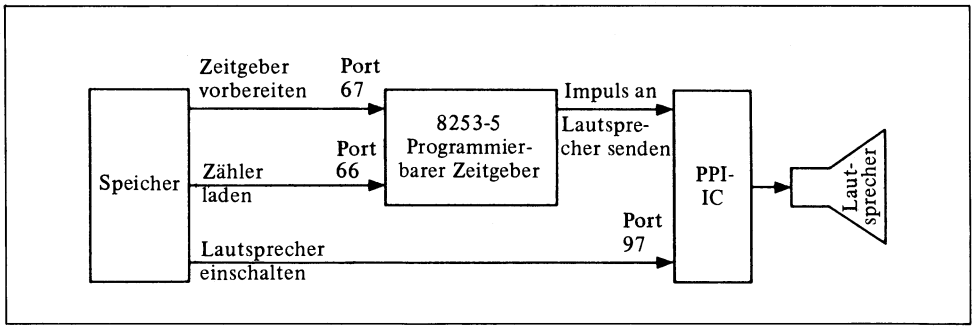


Bild 7-2 Tonerzeugung mit dem 8253-5-Zeitgeber

Bit 0 kontrolliert ein Signal des Zeitgebers, das den Lautsprecher in Betrieb nimmt, Bit 1 kontrolliert das Schwingen des Lautsprechers. Beide Bits müssen auf 1 gesetzt werden, damit eine Übereinstimmung zwischen Lautsprecher und den Steuer-ICs ermöglicht wird. Die Bits werden durch folgendes Programm auf 1 gesetzt, ohne daß die anderen Bits des angesprochenen Bytes verändert werden:

```
70 WERT.ALT = INP (97)            'Wert von Port 97 an WERT.ALT zuweisen
80 WERT.NEU = (WERT.ALT OR &H03) 'Bits 0 und 1 auf den Wert 1 setzen
90 OUT 97, WERT.NEU               'Lautsprecher anschalten
```

7.2.4 Direkte Lautsprecherkontrolle

Der Zeitgeber steuert den Lautsprecher, indem er periodisch Impulse aussendet. Dasselbe kann auch ein Programm durchführen. Setzt man Bit 0 auf den Wert 0 und verändert Bit 1 mit einer festgelegten Geschwindigkeit, so hat dies denselben Effekt wie die Steuerung über das PPI. Je schneller das Programm abläuft, desto höher ist der resultierende Ton. Hier ein Beispiel in BASIC, in dem voraussetzt wird, daß Port 97 (hex 61) den Wert 76 enthält:

```
10 X = INP (97) AND &HFC        'Portwert ändern, die letzten zwei Bits werden auf
                                                    0 gesetzt
20 OUT 98, X                    'Lautsprecher einschalten
30 OUT 97, X + 2                'Lautsprecher ausschalten
40 GOTO 20
```

Die Zeilen 20 und 30 schalten den Lautsprecher ein und aus, zusammen bewirken sie eine volle Schwingung.
Das Beispiel erzeugt den höchstmöglichen Ton, der unter BASIC erreichbar ist. Um die Tonhöhe weiter zu steigern, muß man eine andere Programmiersprache, die schneller als BASIC arbeitet, wählen. Die Erzeugung von klickenden oder summenden Tönen wird durch das Einfügen von Verzögerungen im Programm ermöglicht.
Ungeachtet dieser vielfältigen Möglichkeiten ist die Tonerzeugung mit Hilfe von Programmen nicht die optimale Lösung. Gegenüber der Verwendung des Zeitgebers hat sie drei entscheidende Nachteile:

Ein Programm belastet die CPU, so daß diese keine anderen Aufgaben durchführen kann.

Die Frequenz ist von der Verarbeitungsgeschwindigkeit des Prozessors abhängig. Ein Programm läuft also auf unterschiedlichen Modellen mit einer jeweils anderen Tonhöhe.

Der Taktimpulsinterrupt beeinträchtigt die "Geschmeidigkeit" des Tones, es kommt zum trillern oder schmettern. Lediglich durch Unterdrücken dieses Interrupts kann ein klarer Ton erzeugt werden. Leider verliert der Prozessor dann auch seinen "Zeitsinn".

Es gibt wohl nur einen einzigen Vorteil der direkten Lautsprechersteuerung gegenüber der Zeitgebermethode: Bei direkter Steuerung lassen sich (mit viel Mühe) tatsächlich klare polyphone Töne erzeugen. Der damit verbundete Aufwand wird sich jedoch nur in seltenen Ausnahmefällen lohnen.

7.3 Lautstärke und Tonqualität

Für den internen Lautsprecher der meisten PC-Modelle gibt es keine Lautstärkeregelung. Jeder Lautsprecher spricht aber auf bestimmte Frequenzen besser an als auf andere. Aus diesem einfachen Grund hören Sie manche Töne lauter, andere leiser. Bei solch einfachen Lautsprechern, wie den eingebauten, variiert die Lautstärke bezüglich bestimmter Frequenzen in großem Maße. Mit dem folgenden Programm testen Sie, welche Tonhöhe für Ihre Zwecke die günstigste ist.

```
10 PLAY "MF"                        'jeden Ton separat spielen
20 FREQUENZ = 37
30 WHILE FREQUENZ < 32000           'alle Frequenzen bis 32000 Hertz
40   PRINT USING "##,###";FREQUENZ  'Frequenz anzeigen
50   SOUND FREQUENZ, 5              'Tondauer mit 5 festlegen
60   FREQUENZ = FREQUENZ * 1.1      'Frequenz um 1/10 erhöhen
70 WEND
```

Abgesehen von der Lautstärke unterscheiden sich die Lautsprecher der verschiedenen PC-Modelle auch im Klang. Ein Grund liegt im Gehäuse, das den Lautsprecher völlig umgibt und je nach den Resonanzeigenschaften seines Materials den Ton verändert. Das Timbre eines XT unterscheidet sich von dem des tragbaren PC deutlich, der Standard-PC klingt wieder anders.

Kapitel 8
Grundlegendes über das ROM-BIOS

8.1 Konzeption des BIOS 146
8.2 ROM-BIOS-Interrupts 147
8.3 Grundlegende Eigenschaften der BIOS-Routinen 148
8.4 Erstellen einer Assemblerschnittstellenroutine 149
8.4.1 Grundform einer Schnittstellenroutine 150
8.4.2 Ebene 1: Vereinbarung der Assemblerroutine 151
8.4.3 Ebene 2: Vorbereitung der Assemblerroutine 151
8.4.4 Ebene 3: Ein- und Ausgang der Schnittstelle 152
8.4.5 Ebene 4: Parameterzugriff 153
8.4.6 Ebene 5: Aufruf der ROM-BIOS-Routine 154

Das Geheimnis des erfolgreichen Programmierens auf einem IBM PC oder Kompatiblen liegt in der effizienten Nutzung der eingebauten Software, den ROM-BIOS-Routinen. Diese liegen zwischen der Hardwareebene und höheren Softwareebenen, wozu auch schon das DOS-Betriebssystem zählt. Das ROM-BIOS arbeitet direkt mit der Hardware des Computers und den Peripheriegeräten zusammen. Es erledigt die Grundoperationen wie z.B. das Schreiben einzelner Bytes auf Bildschirm oder Diskette. Die Programmiersprachen und das DOS basieren zu einem großen Teil auf den Grundoperationen, gehen aber entsprechend der jeweiligen Aufgabenstellung darüber hinaus. Wenn Sie professionelle Software schreiben wollen, sei Ihnen die Benutzung der ROM-BIOS-Routinen empfohlen. Sie erhalten damit einen "gut sortierten Werkzeugkasten", der Ihre Arbeit wesentlich erleichtert, so, wie sich die Entwickler des IBM PC das vorgestellt haben.
Der letzte Aspekt bedarf einer zusätzlichen Bemerkung. IBM hat das ROM-BIOS des PC umfangreich ausfallen lassen, um eine exakt festgelegte Steuerung aller Grundoperationen zu ermöglich. Bei der Entwicklung neuer PC-Modelle sind sowohl IBM selbst als auch die Kompatiblenhersteller sorgsam darauf bedacht, ihre ROM-BIOS-Routinen zu früheren Versionen möglichst vollständig kompatibel zu gestalten. Für uns Programmierer bedeutet das: Solange wir unter der direkten oder indirekten Kontrolle der ROM-BIOS-Routinen arbeiten, dürften keine oder zumindest kaum Probleme bezüglich Kompatibilität auftreten. Wer aber das ROM-BIOS umgeht und unmittelbar auf die Hardwareebene zugreift, der fordert Schwierigkeiten geradezu heraus. Zudem beschränkt er seine Software gleichzeitig in der Flexibilität und Portierbarkeit auf andere Computer oder selbst nur auf andere Konfigurationen desselben Rechners.
In den nächsten fünf Kapiteln wollen wir uns näher mit den unterschiedlichen BIOS-Routinen beschäftigen. Glücklicherweise sind die einzelnen Routinen in ihren Aufgaben je nach Peripherie, für die sie zuständig sind, klar voneinander abgegrenzt. Die Bildschirm-, Floppy- und Tastaturroutinen können deshalb gesondert behandelt werden. Bevor wir uns jedoch mit den einzelnen Routinen beschäftigen, sollen Sie erst lernen, wie man sie in ein Programm einfügt. Dazu ist eine grundlegende Erklärung darüber notwendig, wie Schnittstellenprogramme als Brücke zwischen den Programmiersprachen einerseits und den ROM-BIOS-Routinen andererseits aufgebaut werden.

8.1 Konzeption des BIOS

Alle ROM-BIOS-Routinen werden über Interrupts aufgerufen. Zu diesem Zweck steht im unteren Bereich des Hauptspeichers eine Tabelle mit Interruptvektoren zur Verfügung. Ein Interruptvektor ist ein Verweis oder ein Zeiger zur Anfangsadresse einer ROM-Routine. Über diesen Verweis

ist es jedem Programm möglich, über die Vektortabelle auf BIOS-Routinen zuzugreifen, ohne deren tatsächliche Anfangsadresse zu kennen. Angesprochen wird nur der jeweilige Vektor, der auf die Routine zeigt. Dieses Konzept ermöglicht es, im Zuge der Weiterentwicklung der Geräte die BIOS-Routinen zu erweitern und zu verschieben, solange nur die Vektortabelle unverändert bleibt.
Obwohl IBM dennoch bemüht ist, nicht nur die Vektortabelle, sondern auch die absoluten Speicheradressen des ROM-BIOS (die Adressen der Routinen) beizubehalten, wäre es unklug, diese Adressen bei der Programmierung zu benutzen. Durch zukünftige Erweiterungen kann und wird es immer wieder einmal zu Verschiebungen im BIOS kommen. Der beste Weg, eine ROM-Routine aufzurufen, ist die Benutzung der Interrupts, also der indirekte Weg.
Nun wäre es denkbar, den Ablauf von einem einzigen übergeordneten *Master Interrupt* überwachen zu lassen. Das Konzept des IBM PC sieht eine Vielzahl von Interrupts für unterschiedliche Aufgabenbereiche vor. Jeder Bereich hat seinen eigenen Kontrollinterrupt. Der herausragende Vorteil dieser Konzeption liegt auf der Hand: Jede Interruptroutine kann ausgetauscht werden, ohne daß das auf andere Routinen Auswirkungen zeigt. Wird ein neues Gerät auf den Markt gebracht, z.B. ein Bildschirmmonitor oder ein Drucker, kann der betreffende Hersteller eine auf das neue Gerät zugeschnittene Routine mitliefern, die die alte Routine für das entsprechende Gerät ersetzt. Das ROM-BIOS bleibt in allen anderen Teilen vollkommen unverändert. Die neue Routine wird in den meisten Fällen im RAM abgelegt. Diese einfache Möglichkeit zur Erweiterung der Hardware ist gegeben, weil die Interruptstruktur des PC von vornherein nach außen hin offen konzipiert ist.

8.2 ROM-BIOS-Interrupts

Es gibt insgesamt zwölf ROM-BIOS-Interrupts, die sich in fünf Gruppen einteilen lassen. Sechs der zwölf Interrupts bedienen die Peripherie, zwei kontrollieren die Ausstattung des Zentralgerätes, einer arbeitet mit der Zeit/Datum-Uhr, einer übernimmt die Operationen für die Bildschirmausgabe und die letzten beiden bringen den Computer in jeweils andere Arbeitsmodi, sie aktivieren das ROM-BASIC und die Systemstartroutine. Sie werden sehen, daß die meisten der Routinen Zugriff auf eine Anzahl von Unterroutinen nehmen, die die eigentlichen Aufgaben ausführen. Der Bildschirminterrupt verfügt über 16 Unterroutinen, wovon jede eine bestimmte Aufgabe, vom Setzen des Bildschirmmodus bis zur Steuerung der Cursorposition, übernimmt. Um eine bestimmte Routine aufzurufen, wird ihre Nummer in das AH-Register geschrieben. Ein Programmbeispiel finden Sie am Ende des Kapitels.

Interrupt Dez	Hex	Anwendung
Peripherieroutinen		
16	10	Bildschirmroutinen (siehe Kapitel 9)
19	13	Diskettenroutinen (siehe Kapitel 10)
20	14	Kommunikationsroutinen (siehe Kapitel 12)
21	15	Kassettenrekorderroutinen (siehe Kapitel 12)
22	16	Standardtastaturroutinen (siehe Kapitel 11)
23	17	Druckerroutinen (siehe Kapitel 12)
Ausstattungsstatusroutinen		
17	11	Ausstattungsauflistungsroutine (siehe Kapitel 12)
18	12	Hauptspeichergrößenroutine (siehe Kapitel 12)
Zeit/Datum-Routine		
26	1A	Zeit-und-Datum-Routinen (siehe Kapitel 12)
Bildschirmausgaberoutine		
5	5	Bildschirmdruckroutine (siehe Kapitel 12)
Spezialroutinen		
24	18	ROM-BASIC aktivieren (siehe Kapitel 12)
25	19	Systemstartroutine aktivieren (siehe Kapitel 12)

Tabelle 8-1 Die zwölf ROM-BIOS-Interrupts

8.3 Grundlegende Eigenschaften der BIOS-Routinen

Die ROM-BIOS-Routinen verwenden eine Reihe von standardisierten Aufrufkonventionen, um eine weitgehende Konsistenz im Gebrauch der Register, Flaggen, Stapel und des Speichers zu garantieren. Im folgenden sind diese Festlegungen erläutert. Wir beginnen mit den Segmentregistern.
Das Codesegmentregister (CS-Register) wird automatisch als Teil des Interruptprozesses reserviert, geladen und in den Ausgangszustand zurückgesetzt. Wir brauchen uns über dieses Register also keine Gedanken zu machen. Die DS- und ES-Register werden von der jeweils aufrufenden Routine geschützt, außer in einigen wenigen Fällen, wo sie explizit benutzt werden. Das Stapelsegmentregister (SS) bleibt unverändert, denn die ROM-BIOS-Routinen verwenden nur einen Arbeitsstapel, der bereitgestellt werden muß.
Die Anforderungen der einzelnen ROM-BIOS-Routinen in Bezug auf den Stapel können beträchtlich variieren, besonders, da manche Routinen wiederum andere aufrufen. Eine der Schwachstellen des IBM PC und aller anderen Computer, die um den Intel 8088 herum aufgebaut sind, sind die unpräzisen Festlegungen zur Benutzung des Stapels. Sie sollten für Ihre Programme generell einen wesentlich größeren Stapel vorsehen als die BIOS-Routinen, die nur 16 bis 20 Byte benötigen.

In Verbindung mit den Segmentregistern wird der Befehlszähler (PC oder IP) durch denselben Mechanismus wie das Codesegmentregister vor Veränderungen durch die BIOS-Routinen verschont. Der Stapelzeiger (SP) wird insofern geschützt, als daß ihn die BIOS-Routinen nach der Benutzung wieder in den Ausgangszustand zurücksetzen.
Die Register AX bis DX werden durch die meisten BIOS-Routinen geändert. Bei manchen Routinen werden einige Register nicht berücksichtigt, was aber für die Programmierung nicht von Belang ist. Eine Regel läßt sich immerhin aufstellen: Das Resultat einer Routine wird im allgemeinen im AX-Register abgelegt. Dies gilt sowohl für das ROM-BIOS als auch für alle Programmiersprachen.
Für die Indexregister (SI und DI) gilt das Gleiche wie für die Register AX bis DX.
Die Flaggenregister werden als Nebenwirkungen der einzelnen Befehle in den Unterprogrammen verändert. In manchen Fällen werden die Übertragsflagge (CF für *Carry Flag*) oder die Nullflagge (ZF für *Zero Flag*) als Erfolgsmeldung der aufgerufenen Funktion benutzt. Es wäre wohl zu viel verlangt, daß die Flaggenregister bei BIOS-Grundfunktionen unverändert blieben.
Es ließen sich noch mehr Details über Registerveränderungen durch BIOS-Aufrufe anführen. Wir wollen darauf verzichten. Wenn Sie bei der Programmierung den Regeln folgen, die im nächsten Abschnitt aufgestellt werden und sich an die Standards der verwendeten Programmiersprache halten (lesen Sie hierzu auch die Kapitel 19 und 20), sollten Sie gut zurechtkommen. Die BIOS-Routinen vertragen sich mit dem meisten Programmiersprachen, ohne daß bei der Programmierung besondere Vorsichtsmaßnahmen nötig sind.

8.4 Erstellen einer Assemblerschnittstellenroutine

Um die ROM-BIOS-Routinen direkt nutzen zu können, brauchen wir eine Assemblerschnittstellenroutine zwischen Programmiersprache und ROM-BIOS. Bitte beachten Sie, daß das Wort *Schnittstellenroutine* hier ausschließlich im Sinne *Verbindung zwischen BIOS und Programmiersprache* gemeint ist. Eine Schnittstelle wird in Assembler geschrieben, in Objektcode assembliert (.OBJ-Dateien) und in ein anderes Programm (.EXE- oder .COM-Dateien in DOS) eingebunden. Weitere Informationen hierzu können Sie in Kapitel 19 nachlesen.
Das Arbeiten mit Assembler ist eine recht "gefährliche" Sache für jemanden, der damit nicht vertraut ist. Es gibt aber eine Menge guter Gründe, warum man Assembler beherrschen sollte. Eine einfache Schnittstellenroutine zu schreiben ist nicht sehr kompliziert, so daß Sie keinen Grund hätten zu resignieren, falls Sie mit Assembler (noch) nicht sehr vertraut sind.

Um eigene Routinen zu schreiben, brauchen Sie einen Assembler, der die Standards von DOS für Objektdateien erfüllt. Dies kann der "IBM Macro Assembler" sein, aber auch jeder andere auf dem Markt erhältliche. Beachten Sie, daß es einige Assembler gibt, die eher für das Erstellen kompletter Assemblerprogramme als für das Entwickeln von Routinen, die an andere Programme angebunden werden sollen, geeignet sind. Alle Beispiele in diesem Buch beziehen sich auf den "IBM Macro Assembler".

Anmerkung: BASIC kann mit Maschinenspracheunterprogrammen arbeiten, die direkt in den Speicher geladen werden. Im interpretierenden BASIC werden sie mit CALL aufgerufen, im kompilierten BASIC mit CALL ABSOLUTE. Das Vorbereiten einer solchen Routine, die mit BASIC arbeitet, kann mit dem A-Befehl (assemble) von DEBUG mit einem Assembler geschehen.

8.4.1 Grundform einer Schnittstellenroutine

Die Form eines Programmes hängt vom jeweiligen Anwendungszweck ab. Eine Schnittstellenroutine als das Verbindungsstück zwischen der Programmiersprache und dem ROM-BIOS muß auf beide Teile zugeschnitten sein. Es kommt darauf an, welche Programmiersprache verwendet wird, welche BIOS-Routinen angesprochen werden und ob Daten in die eine oder andere Richtung fließen.
Das Gerüst für eine Assemblerschnittstelle zum ROM-BIOS sieht immer gleich aus. Um es zu verstehen, müssen Sie sich mit folgendem Schema vertraut machen:

Ebene 1: Vereinbarung der Assembleroutine
  Ebene 2: Vorbereitung der Assembleroutine
    Ebene 3: Eingang der Schnittstellenroutine
      Ebene 4: Parameterübergabe zur Unterroutine
        Ebene 5: BIOS- oder DOS-Service aufrufen
      Ebene 4: Parameterübergabe zum aufrufenden Programm
    Ebene 3: Ausgang der Schnittstellenroutine
  Ebene 2: Nachspann der Unterroutine
Ebene 1: Assemblernachspann

In diesem Konzept werden die Ebenen 1 und 2 benötigt, um die nötigen Vorbereitungen zu treffen, es werden aber noch keine Befehle im Sinne der tatsächlichen Aufgabe abgearbeitet. Die Ebenen 3 bis 5 übernehmen die Ausführung der eigentlichen Maschinenprachebefehle.
Beachten Sie bitte, daß unterschiedliche Umstände die spezifischen Anforderungen der Interruptroutine verändern können. In diesem Kapitel finden Sie diejenigen Elemente, die für alle Routinen verbindlich sind.

8.4.2 Ebene 1: Vereinbarung der Assemblerroutine

Nachfolgend sehen Sie einen typischen Ausschnitt der Ebene 1 einer Schnittstellenroutine. Die Zahlen sind lediglich Bezugspunkte für die nachfolgenden Erklärungen und ansonsten ohne Bedeutung.

```
SCHNITTSTELLE SEGMENT 'CODE'          ;     1-1
              ASSUME CS:SCHNITTSTELLE ;     1-2
; Die Ebenen 2 bis 5 stehen hier.
SCHNITTSTELLE ENDS                    ;     1-3
              END                     ;     1-4
```

SCHNITTSTELLE ist ein eigenständiger Name in Zeile 1-1, den wir dieser Assemblerroutine gegeben haben; SEGMENT ist ein Befehlswort, das zur Definition einer Assemblerroutine benutzt wird; CODE gehört in eine Kategorie, die je nach Anwendung variieren kann, wie wir noch in einem anderen Beispiel sehen werden.
Die Zeile 1-2 wird nicht immer benötigt, Assemblerexperten werden schon festgestellt haben, daß sie eigentlich ein "unerlaubter Schwindel" ist. Der ASSUME-Befehl erlaubt es uns aber, IF-THEN-Befehle ohne weitere Probleme zu benutzen. In späteren Kapiteln kommen wir darauf zurück.
Zeile 1-3 beendet den SEGMENT-Befehl, der in Zeile 1-1 aufgerufen wurde, die Zeile 1-4 schließt die Assemblerroutine ab.
Die Formatvereinbarungen, die hier zum Tragen kommen, sind aus IBM/ Microsoft Pascal übernommen. Mehr Informationen über Pascal finden Sie in Kapitel 20. Eine leicht veränderte Version des obigen Programms erfordert die Programmiersprache C: Anstelle der Zeile 1-1 müssen die folgenden zwei Zeilen eingegeben werden.

```
PGROUP     GROUP     PROG
    SCHNITTSTELLE SEGMENT   BYTE PUBLIC 'PROG'
```

8.4.3 Ebene 2: Vorbereitung der Assemblerroutine

Hier sehen Sie einen typischen Ausschnitt aus Ebene 2, der *Prozedur* genannt wird:

```
         PUBLIC  MEMSIZE          ;     2-1
MEMSIZE  PROC    FAR              ;     2-2
; Die Ebenen 3 bis 5 stehen hier.
MEMSIZE  ENDP                     ;     2-3
```

Die Zeile 2-1 weist den Assembler an, den Namen der Prozedur, MEMSIZE, als allgemein zugänglich zu deklarieren. Das Bindeprogramm (Linker) ist nun in der Lage, die Routine an aufrufende Programme anzuhängen.

Die Zeile 2-2 und 2-3 stellen eine Art Klammerung um die Prozedur dar. PROC und ENDP sind vorgeschrieben und umschließen jede Prozedur, wobei PROC den Beginn und ENDP das Ende der Prozedur definiert. FAR zeigt dem Assembler an, daß die Prozedur außerhalb des momentan angesprochenen Segmentes liegt. Wir können an dieser Stelle entweder FAR oder NEAR verwenden. NEAR bedeutet, daß die angesprochene Routine innerhalb des aktuellen Segmentes liegt. FAR-Aufrufe sind weit häufiger anzutreffen, einige Programmiersprachen müssen (C) oder können (Pascal) NEAR-Aufrufe verwenden. Abgesehen von NEAR und FAR ist dieses Beispiel auf alle Sprachen und Anwendungen übertragbar.

8.4.4 Ebene 3: Ein- und Ausgang der Schnittstelle

Die Ebene 3 ist dafür verantwortlich, daß die Unterroutine und die aufrufende Programmiersprache einander vertragen und sich nicht gegenseitig stören. Ein Beispiel:

```
PUSH    BP      ;       3-1
MOV     BP,SP   ;       3-2

; Die Ebenen 4 und 5 stehen hier.

POP     BP      ;       3-3
RET     0       ;       3-4
```

Die Zeilen 3-1 und 3-2 ermöglichen einen Zugriff auf die vom aufrufenden Programm übernommenen Parameter und schützen sie gleichzeitig. Für die Verwaltung des Stapeleingangs wird generell das BP-Register verwendet. Das aufrufende Programm hat ein eigenes BP-Register, das in Zeile 3-1 in den Stapel geschoben wird (PUSH), um es zu schützen. In Zeile 3-3 wird es mit dem POP-Befehl wieder hervorgeholt.
In Zeile 3-2 wird ein eigener Stapel für die Routine angelegt, indem der Stapelzeiger (SP) nach BP transferiert wird. Müssen bestimmte Register für das aufrufende Programm geschützt werden, sind sie direkt nach Zeile 3-2 in den Stapel zu schieben (mit PUSH) und direkt vor Zeile 3-3 mit dem POP-Befehl wieder herauszuholen. Im Regelfall ist dies jedoch nicht notwendig.
Die Zeile 3-4 übergibt die Kontrolle von der Routine zurück an das aufrufende Programm. Der Assembler übersetzt den RET-Befehl in NEAR oder FAR, abhängig davon, ob die Prozedur PROC als NEAR oder FAR deklariert wurde. Zeile 3-4 übernimmt die "Aufräumarbeit", die normalerweise immer notwendig ist, um die Parameter aus dem Stapel zu entfernen. Sind keine Parameter vorhanden oder nimmt das aufrufende Programm die Parameter vom Stapel, wie das z.B. in der Sprache C der Fall ist, wird dieser Wert 0 (wie in unserem Beispiel). Mehr über die Programmiersprache C finden Sie in Kapitel 20. Sind Parameter vorhanden und

wird durch die Programmiersprache der Stapel nicht in den Ursprungszustand zurückgesetzt, dann muß die Anzahl der Parameter bekannt sein. Der Wert in Zeile 3-4 muß für jeden 1- oder 2-byte-Parameter (Byte, Wort oder Offsetadresse) um zwei erhöht werden, für jeden 4-byte-Parameter (segmentierte Adresse) um vier. Solange Sie Ihre Programme exakt kennen, stellt dies kein Problem dar. Nur die vier oben genannten Elemente kommen als Parameter in Frage.

8.4.5 Ebene 4: Parameterzugriff

Die Ebene 4 behandelt die Weitergabe der Parameter vom aufrufenden Programm zum ROM-BIOS und umgekehrt die Übergabe der Ergebnisse vom ROM-BIOS an das aufrufende Programm. Die Parameter des Programmes befinden sich als Daten oder Adressen im Stapel; Details stehen in Kapitel 20. Meist werden die Register AX bis DX für die Ein- und Ausgabe des ROM-BIOS verwendet.
Die Routine findet die Parameter im Stapel, indem sie relativ zur BP-Festlegung adressiert. Hier eine typische Anordnung:

Position	**Inhalt**
BP	gesicherter BP des aufrufenden Programmes
BP + 2	Rücksprungadresse, Offset und Segment
BP + 6	erster Parameter
BP + 8	zweiter Parameter
BP + 10	dritter Parameter

Die Rücksprungadresse an der Position BP + 2 ist vier Bytes für eine FAR-Prozedur, aber nur zwei Bytes für eine NEAR-Prozedur, lang. Wenn Sie eine NEAR-Prozedur aufrufen, verschieben sich alle folgenden Positionen um minus zwei, d.h., aus BP + 6 wird BP + 4 usw. Die meisten Sprachen schieben ihre Parameter in der Reihenfolge des Schreibens in den Stapel, der letzte Parameter belegt somit die Spitze des Stapels, das wäre hier BP + 6. Lattice/ Microsoft C arbeitet mit der umgekehrten Reihenfolge, so daß der erste im aufrufenden Programm geschriebene Parameter an der Stapelspitze steht.
Parameter belegen entweder zwei oder vier Bytes im Stapel, zwei Bytes ist der häufigere Fall. In unserem Beispiel sind die Positionen BP + 6, BP + 8 und BP + 10 jeweils zwei Bytes auseinander. Würde ein 4-byte-Parameter auftauchen, müßten ab dieser Stelle alle folgenden Positionen entsprechend geändert werden.
Befinden sich Daten (im Unterschied zu Adressen) im Stapel, kann man sie wie folgt aufrufen: [BP + 6]. Wenn hingegen eine Adresse im Stapel steht, sind zwei Schritte notwendig, mit dem ersten erhält man die Adresse, mit dem zweiten die Daten. Unser Beispiel zeigt Ihnen beide Möglichkeiten auf:

```
MOV     AX,[BP+6]        ; Wert des Parameters 1          4-1
MOV     DX,[BP+8]        ; Adresse des Parameters 2       4-2
MOV     BX,[DX]          ; Wert des Parameters 2          4-3
; Hier steht Ebene 5.
MOV     DX,[BP+8]        ; Adresse des Parameters 2       4-4
MOV     [DX],BX          ; Gibt neuen Wert zurück         4-5
```

Die MOV-Befehle transferieren die Daten vom zweiten in den ersten Operanden. Zeile 4-1 holt den Parameter 1 aus dem Stapel und speichert ihn in das AX-Register. Die Zeilen 4-2 und 4-3 holen einen Parameter über seine Adresse aus dem Stapel, Zeile 4-2 speichert die Adresse des Parameters im DX-Register und Zeile 4-3 verwendet die Adresse, um den tatsächlichen Wert zu holen und im BX-Register abzulegen. Die Zeilen 4-4 und 4-5 kehren diesen Prozeß um, die Zeile 4-4 holt erneut den Adreßwert und die letzte Zeile (4-5) bringt den Inhalt des BX-Registers auf die vorher angesprochene Speicherstelle.

Anmerkung: Hier wird zugleich ein grundlegender Aspekt der Assemblernotation angesprochen: "AX" bezieht sich auf den Wert, der in AX steht und "[AX]" bezieht sich auf eine Speicherstelle, deren Adresse in AX steht. Die Unterscheidung zwischen einem Wert und seiner Adresse müssen Sie stets beachten.

8.4.6 Ebene 5: Aufruf der ROM-BIOS-Routine

Die Ebene 5 ist der letzte Schritt, nämlich der Aufruf der BIOS-Routine. Dazu sind zwei Befehle erforderlich:

```
MOV   AH,1    ;Funktion 1  5-1
INT   16      ;BIOS-Aufruf 5-2
```

In Zeile 5-1 wird eine Interruptroutine ausgewählt. Die Routinen sind von 0 an aufsteigend durchnumeriert und werden angesprochen, indem der entsprechende Wert in das AH-Register eingeschrieben wird.
Zeile 5-2 erzeugt den Interrupt, der die BIOS-Routine aufruft, in unserem Beispiel ist das der Interrupt 16 (hex 10), der Bildschirminterrupt.
Das Modell mit fünf Ebenen zeigt nahezu alle Aspekte einer Assemblerschnittstelle zu BIOS-Routinen. In den folgenden Kapiteln werden Sie anhand von Beispielen sehen, wie dieses Schema zur Anwendung gelangt.

Kapitel 9
Bildschirmroutinen im ROM-BIOS

9.1 Zugriff auf die Bildschirmroutinen 156
9.1.1 Routine 0: Bildschirmmodus festlegen 157
9.1.2 Routine 1: Cursorgröße festlegen 158
9.1.3 Routine 2: Cursorposition setzen 159
9.1.4 Routine 3: Cursorposition abfragen 159
9.1.5 Routine 4: Lichtgriffelposition abfragen 160
9.1.6 Routine 5: Aktive Anzeigeseite festlegen 160
9.1.7 Routine 6: Fenster nach oben rollen 161
9.1.8 Routine 7: Fenster nach unten rollen 162
9.2 Zeichenhandhabungsroutinen 162
9.2.1 Routine 8: Zeichen und Attribut an der Cursorposition lesen 162
9.2.2 Routine 9: Zeichen und Attribut an der Cursorpositon schreiben 163
9.2.3 Routine 10: Zeichen an der Cursorposition schreiben 164
9.3 Grafikroutinen 165
9.3.1 Routine 11: Farbpalette festlegen 165
9.3.2 Routine 12: Pixel setzen 166
9.3.3 Routine 13: Pixel abfragen 166
9.3.4 Routine 14: TTY-Zeichen schreiben 167
9.3.5 Routine 15: Bildschirmmodus feststellen 167
9.3.6 Routine 19: Zeichenkette (String) schreiben 168
9.4 Anmerkungen und Beispiel 168

In diesem Kapitel werden alle Video- oder Bildschirmroutinen des ROM-BIOS behandelt. Der größte Teil des Kapitels ist der Beschreibung der Routinen gewidmet. Am Ende folgen einige Programmierhinweise und eine Assemblerroutine als Anwendungsbeispiel. Die verschiedenen Bildschirmmodi wurden ausführlich in Kapitel 4 behandelt. Informationen über die vom BIOS verwendeten Speicherstellen finden Sie in Kapitel 3.2.2.

9.1 Zugriff auf die Bildschirmroutinen

Die ROM-BIOS-Bildschirmroutinen sind von 0 bis 15 durchnumeriert und werden alle mit dem Interrupt 16 (hex 10) aufgerufen. Beim AT findet man eine zusätzliche Routine. Die Auswahl einer Routine erfolgt, indem ihre Nummer in das AH-Register geschrieben wird. Meist werden noch Parameter übergeben, die den Anforderungen der BIOS-Routine entsprechend und für gewöhnlich in den Registern BX, CX und DX stehen müssen. Zweck und Plazierung der Parameter werden wir in jedem Abschnitt besprechen.

Routine Dez	Routine Hex	Beschreibung
0	0	Bildschirmmodus festlegen
1	1	Cursorgröße festlegen
2	2	Cursorposition setzen
3	3	Cursorposition abfragen
4	4	Lichtgriffelposition abfragen
5	5	Aktive Anzeigeseite festlegen
6	6	Fenster nach oben rollen
7	7	Fenster nach unten rollen
8	8	Zeichen und Attribut an der Cursorposition lesen
9	9	Zeichen und Attribut an der Cursorposition schreiben
10	A	Zeichen an der Cursorposition schreiben
11	B	Farbpalette festlegen
12	C	Pixel setzen
13	D	Pixel abfragen
14	E	TTY-Zeichen schreiben
15	F	Bildschirmmodus feststellen
19	13	Zeichenkette (String) schreiben

Tabelle 9-1 Die 17 Bildschirmroutinen des ROM-BIOS

9.1.1 Routine 0: Bildschirmmodus festlegen

Die Routine 0 wird benutzt, um einen der zwölf Bildschirmmodi auszuwählen, die detailliert in Kapitel 4.2 besprochen wurden.
Wie Sie den Erläuterungen in Kapitel 4 entnehmen konnten, sind die Modi 0 bis 6 nur zusammen mit dem Farb-/Grafikadapter einsetzbar, der Modus 7 ist der Monochrommodus und die Modi 10 und 13 bis 15 können nur mit der HR-Farbgrafikkarte EGA (*Enhanced Graphics Adapter*) verwendet wurden. Die Modi 8 und 9 sind nur auf dem PCjr (der in Deutschland nie zur Auslieferung kam) verfügbar und werden aus diesem Grunde hier nicht weiter erwähnt.
Sie sollten außerdem wissen, daß die Schwarz/Weiß- und Farbunterdrükkungsmodi (Modi 0, 2 und 5) die Farben nur bei Benutzung des Mischsignalausgangs unterdrücken, nicht beim RGB-Ausgang.
Normalerweise wird der Bildschirmpuffer beim Aufruf eines Modus gelöscht; das gilt sogar dann, wenn kein Moduswechsel stattfindet. Das wiederholte Setzen desselben Modus ist ein einfacher und effektiver Weg, den Bildschirm zu löschen.

Modus	Art	Farben	Auflösung	Bildschirmpunkte	Adapter
0	Text	S/W	Mittel	40 X 25	FGA
1	Text	16	Mittel	40 X 25	FGA
2	Text	S/W	Hoch	80 X 25	FGA
3	Text	16	Hoch	80 X 25	FGA
4	Grafik	4	Mittel	320 X 200	FGA
5	Grafik	4 Grau	Mittel	320 X 200	FGA
6	Grafik	S/W	Hoch	640 X 200	FGA
7	Text	S/W	Hoch	80 X 25	MA
10	Grafik	4, 64	Hoch	640 X 200	EGA
13	Grafik	16	Mittel	320 X 200	EGA
14	Grafik	16	Hoch	640 X 200	EGA
15	Grafik	4	Sehr hoch	640 X 350	EGA

Erläuterungen:

FGA steht für Farb-/Grafikadapter; MA bedeutet Monochromadapter; EGA ist die Abkürzung für „Enhanced Graphics Adapter" (HR-Farbgrafikadapter).

Tabelle 9-2 Festlegung des Bildschirmmodus mit Routine 0

In Kapitel 4 finden Sie mehr über die Bildschirmmodi, in Kapitel 3.2.2 können sie sich eine nähere Beschreibung der Speicherstelle hex 449 näher ansehen, um zu erfahren, wie die Bildschirmmodi gespeichert werden. Wie der aktuelle Bildschirmmodus gelesen werden kann, erfahren Sie in Kapitel 9.3.5 (Bildschirmroutine 15, hex F).

9.1.2 Routine 1: Cursorgröße festlegen

Die Routine 1 beeinflußt Form und Größe des in den Textmodi auftretenden Cursor. Der standardisierte IBM Cursor erscheint gewöhnlich ein oder zwei Rasterzeilen stark am unteren Ende des Zeichenfeldes (das ist das Feld, das für ein Zeichen zur Verfügung steht) und blinkt. Wir können die Cursorgröße verändern, indem wir die Anzahl der angezeigten Rasterzeilen umdefinieren.
Der Farb-/Grafikadapter kann maximal einen Cursor mit der Höhe von acht Rasterzeilen anzeigen, die von 0 (oberste Rasterzeile) bis 7 (unterste Rasterzeile eines Zeichenfeldes) durchnumeriert sind. Der Monochromadapter verfügt über 14 Rasterzeilen, die ebenfalls von oben (0) nach unten (13) durchnumeriert werden. Die Cursorgröße läßt sich verändern, indem Anfangs- und Endrasterzeile neu festgelegt werden, ähnlich wie beim LOCATE-Befehl in BASIC. Die Nummer der Anfangsrasterzeile wird im Register CH abgespeichert, die der Endrasterzeile in CL. Standardmäßig weisen die Register beim Farb-/Grafikadapter folgende Werte auf: CH = 6, CL = 7. Für den Monochromadapter gilt: CH = 12, CL = 13. Ist der Wert in CH niedriger als der in CL, erscheint auf dem Bildschirm ein normaler einteiliger Cursor, andernfalls schlägt der Cursor "hinter dem Bildschirm" herum und erscheint sozusagen am oberen Rand des Zeichenfeldes wieder. Der Cursor ist "zweiteilig" geworden.
Sie haben vielleicht bemerkt, daß die gültigen Nummern der Rasterzeilen nur drei Bits (Bit 0 bis 2) des Registers ausfüllen. Wird Bit 5 des Registers CH auf 1 gesetzt, was einem Wert von 32 oder hex 20 entspricht, verschwindet der Cursor vom Bildschirm. Ist ein Grafikmodus eingeschaltet, wird dieses Bit automatisch gesetzt, damit der Cursor nicht die Grafik stört. Dies ist eine der zwei Methoden, wie man im Textmodus den Cursor am Bildschirm unterdrücken kann. Eine andere Möglichkeit besteht darin, den Cursor außerhalb des Bildschirmes zu positionieren, z.B. in Zeile 26, Spalte 1. Obwohl es in den Grafikmodi keinen "echten" Cursor gibt, kann doch durch Abbilden eines Blockzeichens (CHR$(223)) oder durch Verändern der Hintergrundattribute eine Cursorsimulation erzeugt werden.

Routinennummer	Parameter
AH = 1	CH = Anfangsrasterzeile des Cursor CL = Endrasterzeile des Cursor

Tabelle 9-3 Die Register zur Festlegung der Cursorgröße mit Hilfe der Routine 1

Mehr über den Cursor finden Sie in Kapitel 4.8. Die Routine 3 liest die Cursorposition, informieren Sie sich auch dort.

9.1.3 Routine 2: Cursorposition setzen

Die Routine 2 setzt die Position des Cursor mit Hilfe eines Koordinatensystems aus Zeilen und Spalten. In den Textmodi, in denen der Speicher in mehrere Bildschirmseiten aufgeteilt sein kann, wird für jede Seite eine eigene Cursorposition gespeichert. Obwohl die Grafikmodi keinen sichtbaren Cursor haben, wird auch bei ihnen eine logische Cursorposition auf dieselbe Weise wie in den Textmodi gespeichert. Diese logische Cursorposition findet zur Überwachung der Ein- und Ausgabe von Zeichen Verwendung.
Die Cursorposition wird durch das Setzen der gewünschten Zeile in Register DH, der Spalte in DL und der Seite in BH bestimmt. Die Numerierung der Zeilen und Spalten beginnt in der linken oberen Ecke mit den Koordinaten (0,0) und setzt sich in den Textmodi zeilenweise fort. Auch die Grafikmodi verwenden überwiegend die Zeichenkoordinaten, selten die Koordinaten der Pixel. Die Seitennumerierung entspricht der in BASIC verwendeten. In den 40-Spalten-Modi gibt es die Seiten 0 bis 7, in den 80-Spalten-Modi die Seiten 0 bis 3. In einem Grafikmodus muß die Seitennummer immer auf 0 gesetzt werden.

Routinennummer	Parameter
AH = 2	DH = Zeilennummer DL = Spaltennummer BH = Seitennummer (in den Grafikmodi = 0)

Tabelle 9-4 Die Register zum Setzen der Cursorposition mit Routine 2

Mehr über Bildschirmseiten finden Sie in Kapitel 4.4.1. Dort stehen auch Details über Textanzeigeformate. Die Routine 3 liest die Cursorposition.

9.1.4 Routine 3: Cursorposition abfragen

Die Routine 3 ist das Gegenstück zu den Routinen 1 und 2. Wenn wir eine Seitennummer in Register BH spezifizieren und Routine 3 aufrufen, meldet BIOS die Cusorposition in DH (Zeile) und DL (Spalte). Außerdem steht die Anfangsrasterzeile in CH und die Endrasterzeile in CL. Wie bei der Routine 2 muß auch hier die Seitennummer in den Grafikmodi auf 0 gesetzt werden.
Erläuterungen zu den Bildschirmseiten finden Sie in Kapitel 4.4.1, die Textanzeigeformate werden dort ebenfalls behandelt.

Routinennummer	Parameter
AH = 3	BH = Seitennummer (0 in Grafikmodi) DH = Zeilennummer DL = Spaltennummer CH = Anfangsrasterzeile des Cursor CL = Endrasterzeile des Cursor

Tabelle 9-5 Die verwendeten Register zum Lesen der Cursorposition mit Routine 3

9.1.5 Routine 4: Lichtgriffelposition abfragen

Die Routine 4 gibt an, ob der Lichtgriffel ausgelöst wurde (Register AH = 1) oder nicht (Register AH = 0). Das Pixel, auf dem der Lichtgriffel steht, wird durch die Hardware überwacht. Das ROM-BIOS meldet die Position auf zwei Arten: die Zeichenposition mit der Zeile in DH und der Spalte in DL und die Pixelposition mit der Rasterzeile in CH und der Rasterspalte in BX. Da in einer Zeile mehr als 255 Pixel stehen können, wird BX als 2-byte-Register benutzt, alle andere Werte werden in jeweils einem Byte dargestellt.

Routinennummer	Parameter
AH = 4	DH = Zeilennummer (Zeichen) DL = Spaltennummer (Zeichen) CH = Rasterzeile des Pixel (0 bis 199) BX = Spaltennummer des Pixel

Tabelle 9-6 Die von Routine 4 verwendeten Register zum Lesen der Lichtgriffelposition

9.1.6 Routine 5: Aktive Anzeigeseite festlegen

Routine 5 bestimmt die aktive Anzeigeseite im Textmodus. Für die Modi 0 bis 3 wird die Seitennummer im Register AL gespeichert. In einem 40-Spalten-Modus kann zwischen den Seiten 0 bis 7 gewählt werden, in einem 80-Spalten-Modus zwischen den Seiten 0 bis 3. Ohne eine Angabe wird Seite 0 angezeigt. Diese ist am Anfang des Speichers plaziert, die jeweils nächste Seite beginnt 2 Kbyte höher im Speicher (für 40-Spalten-Modi) oder 4 Kbyte höher (für 80-Spalten-Modi). Je höher die Seitennummer, desto höher ist auch die Speicherposition.

Routinennummer	Parameter
AH = 5	AL = Seitennummer der ab nun anzuzeigenden Seite (0–3 für die Modi 2 und 3, 0–7 für die Modi 0 und 1)

Tabelle 9-7 Die von Routine 5 verwendeten Register zur Festlegung der aktiven Anzeigeseite

In Kapitel 4.4.1 finden Sie mehr über Anzeigeseiten.

9.1.7 Routine 6: Fenster nach oben rollen

Die Routinen 6 und 7 werden zur Definition von rechteckigen Textfenstern verwendet, deren Inhalte eine oder mehrere Zeilen nach oben oder unten *gerollt* werden können (*Scrolling*). Um diesen Rolleffekt zu erzeugen, werden am unteren Rand (Routine 6) oder am oberen Rand (Routine 7) Leerzeilen eingeschoben. Die Zeilen werden sozusagen nach oben (Routine 6) bzw. nach unten (Routine 7) aus dem Bild "gerollt".
Die Anzahl der zu rollenden Zeilen wird im Register AL spezifiziert. Steht in AL der Wert 0, ist das gesamte Fenster leer; dasselbe passiert auch, wenn wir mehr Zeilen rollen, als das Fenster groß ist. Die Position bzw. Größe des Fensters wird in den Registern CX und DX abgelegt. CH enthält die oberste Zeile, DH die unterste. CL beinhaltet die linke Spalte, DL die rechte. Das Anzeigeattribut für die Leerzeilen, die eingeschoben werden sollen, wird dem Register BH entnommen.

Routinennummer	Parameter
AH = 6	AL = Anzahl der zu rollenden Zeilen CH = Zeilennummer der oberen linken Ecke CL = Spaltennummer der oberen linken Ecke DH = Zeilennummer der unteren rechten Ecke DL = Spaltennummer der unteren rechten Ecke BH = Anzeigeattribut der Leerzeilen

Tabelle 9-8 Die Register, die von den Routinen 6 und 7 zum Definieren eines Fensters und dem Rollen des Inhaltes verwendet werden

Das Rollen des Fensters ist ein zweiphasiger Prozeß. Wenn eine neue Zeile für das Fenster vorbereitet ist, wird im ersten Schritt die Routine 6 bzw. 7 aufgerufen, um den Inhalt des Fensters zu verschieben. Der zweite Schritt besteht darin, mit Hilfe der Routinen zur Cursorpositionierung und zum Schreiben von Zeichen die neue Zeile hinzuzufügen.

Das folgende Beispiel verdeutlicht das Arbeiten mit Fenstern:

DEBUG	DEBUG aufrufen (von DOS aus)
A	Assemblereingabe
INT 10	Interrupt hex 10
(RETURN)	Assemblierung beenden
R AX	Inhalt von AX einsehen, um ihn zu ändern
0603	Routine 6 (Hochrollen) wählen, Fenster mit drei Zeilen
R CX	Inhalt von CX einsehen, um ihn zu ändern
050A	Obere linke Ecke: Zeile 5, Spalte 10
R DX	Inhalt von DX einsehen, um ihn zu ändern
1020	Rechte untere Ecke: Zeile 16, Spalte 32
D 0 L 180	Bildschirm füllen
G = 100 102	INT 10 durchführen

In Kapitel 8 finden Sie mehr über Assemblerroutinen. Nähere Erläuterungen zu DEBUG entnehmen Sie bitte dem DOS-Handbuch der IBM.

9.1.8 Routine 7: Fenster nach unten rollen

Die Routine 7 ist das genaue Spiegelbild der Routine 6; der Unterschied besteht darin, daß bei Routine 7 die Leerzeilen am oberen Ende des Fensters erscheinen und die alten Zeilen nach unten aus dem Bild verschwinden, bei Routine 6 ist es genau umgekehrt. Die Erläuterung der Parameter lesen Sie bitte unter Routine 6 nach.

9.2 Zeichenhandhabungsroutinen

9.2.1 Routine 8: Zeichen und Attribut an der Cursorposition lesen

Die Routine 8 kann Zeichen "aus dem Bildschirm", d.h., direkt aus dem Bildschirmspeicher lesen. Das Bemerkenswerte an der Routine ist, daß sie sowohl im Textmodus als auch im Grafikmodus eingesetzt werden kann.
In den Grafikmodi werden dieselben Zeichenformtabellen, die in den Textmodi zur Ausgabe von Zeichen dienen, zur Zeichenerkennung benutzt, indem der Bildschirminhalt mit den Zeichenmustern verglichen wird (*Pattern Matching*). Sogar selbstdefinierte Zeichen werden erkannt. In den Textmodi sind die ASCII-Zeichencodes direkt aus dem Bildschirmspeicher ablesbar.

Routinennummer	Parameter
AH = 8	BH = Aktive Anzeigeseite (wird im Grafikmodus nicht benötigt) AL = ASCII-Zeichen, das sich an der Position des Cursor befindet AH = Attribut des Textzeichens

Tabelle 9-9 Die Register, die von Routine 8 verwendet werden, um ein Textzeichen und dessen Attribut zu lesen

Die Routine 8 holt den ASCII-Zeichencode des Zeichens vom Bildschirm und legt ihn in Register AL ab. Wird im Grafikmodus ein Zeichen erkannt, das nicht dem Standard-ASCII-Zeichensatz angehört, steht in AL der Wert 0. In den Textmodi muß die Routine ebenfalls die Textfarbenattribute in BH ablegen. Im Grafikmodus ist das Festlegen einer Anzeigeseite nicht nötig.
In Kapitel 4.3.2 finden Sie mehr über Textzeichen und deren Attribute. Text- und Grafikmoduszeichen werden in Kapitel 4.4.3 erläutert. Anhang C gibt Ihnen Informationen über die ASCII-Zeichen.

9.2.2 Routine 9: Zeichen und Attribut an der Cursorposition schreiben

Die Routine 9 schreibt ein oder mehrere identische Zeichen mit den Farbattributen auf den Bildschirm. Das Zeichen wird in AL spezifiziert, das Attribut des Textmodus oder die Farbe im Grafikmodus werden in BL abgelegt. Wie oft das Zeichen geschrieben werden soll (einmal oder öfter), wird in CX festgelegt.
In den Textmodi muß in Register BH die Anzeigeseite angegeben werden, in den Grafikmodi ist dies nicht nötig.
Das Zeichen wird mit den Attributen ab der momentanen Cursorposition so oft wie in CX spezifiziert geschrieben. Obwohl der Cursor nicht bewegt wird, erscheinen die Zeichen in der Reihenfolge der Bildschirmpositionen. Falls beim Schreiben das Zeilenende überschritten wird, wechselt die Routine in den Textmodi in die nächstfolgende Zeile. In den Grafikmodi entfällt diese nützliche Arbeitsweise.
Die Routine eignet sich für die Darstellung eines einzelnen Zeichens ebenso wie für das Füllen eines Bildschirmbereiches, z.B. mit Leerzeichen. Wenn nur ein Zeichen auf den Bildschirm gebracht werden soll, muß in CX eine 1 stehen. Der Wert 0 in diesem Register führt zu unendlich vielen Zeichen!
Die Routine 9 hat einen Vorteil gegenüber der Routine 14: Man kann die Farbattribute steuern. Leider ist auch ein Nachteil zu verzeichnen: Der Cursor wird nicht automatisch mitbewegt.

Routinennummer	Parameter
AH = 9	AL = ASCII-Zeichen, das auf den Bildschirm geschrieben werden soll BL = Zeichenattribut BH = Aktive Anzeigeseite (kann in den Grafikmodi entfallen) CX = Anzahl der Duplikate des Zeichens

Tabelle 9-10 Die von Routine 9 benutzten Register zum Schreiben eines Textzeichens und seiner Attribute

Für die Grafikmodi gilt: Die in BL spezifizierte Farbe ist die Vordergrundfarbe – die Farbe der Pixel, die das Zeichen darstellen. Ist Bit 7 auf 1 gesetzt, werden die Farb-Bits in BL mit der momentan gültigen Pixel-Farbe mit einer *Exklusiv-Oder*-Operation (XOR) verknüpft. Das ist eine geläufige Methode, um sicherzustellen, daß die resultierende Farbe sich von der vorherigen unterscheidet. Ist Bit 7 gleich 0, wird die Farbe einfach ausgetauscht. Dies gilt auch für die Zeichen- und Pixel-Schreibroutinen 10 und 12.
In Kapitel 4.3.2.1 finden Sie mehr über Anzeigeattribute in den Textmodi, in Kapitel 4.3.2.3 können Sie alles über Farbattribute in den Grafikmodi nachlesen.

9.2.3 Routine 10 (hex A): Zeichen an der Cursorposition schreiben

Die Routine 10 stimmt mit der Routine 9 bis auf eine Ausnahme völlig überein. Während es mit Routine 9 erlaubt ist, existierende Bildschirmfarbattribute im Textmodus zu verändern, ist dies mit Routine 10 nicht möglich.

Routinennummer	Parameter
AH = 10	AL = ASCII-Zeichen, das geschrieben werden soll BL = Farbattribut in den Grafikmodi BH = Aktive Anzeigeseite CX = Anzahl der Duplikate des Zeichens

Tabelle 9-11 Die von Routine 10 verwendeten Register zum Schreiben eines Zeichens

In den Grafikmodi muß die Farbe spezifiziert werden, was dem Charakter einer reinen Zeichenroutine widerspricht. Für die Benutzung der Farben in den Grafikmodi gelten dieselben Regeln wie bei den Routinen 9 und 12: Die Farbe kann direkt oder mit der Operation *Exklusiv-Oder* definiert werden. Nähere Erläuterungen finden Sie unter der Besprechung der Routine 9.

Auch hier sei der Hinweis auf Kapitel 4.3.2.1 gegeben. Dort erfahren Sie mehr über die Anzeigeattribute in den Textmodi. In Kapitel 4.3.2.3 finden Sie Informationen zu den Farbattributen.

9.3 Grafikroutinen

9.3.1 Routine 11 (hex B): Farbpalette festlegen

Die Routine 11 wird zur Auswahl einer der beiden Farbpaletten (mittlere Auflösung) verwendet. Um die Routine zu nutzen, laden Sie das Register BH mit dem Farbpalettenwert und BL mit einem Farbwert.
In den Textmodi gibt es nur ein Einsatzgebiet für diese Routine: Wird BH auf 0 gesetzt, spezifiziert BL die Farbe einer Umrandung des Textes, die aus der gesamten, 16 Farben umfassenden, Palette ausgewählt werden kann. In den Grafikmodi spezifiziert BL die standardmäßige Hintergrundfarbe und die Farbe der Umrandung, wenn BH gleich Null ist. Die Randfarbe verschmilzt mit jedem Bereich des Bildschirmes, der auf die Hintergrundfarbe gesetzt wird. Der Wert für BL kann aus der 16-Farbenpalette ausgewählt werden.
Ist BH gleich 1, wird in BL die aktuelle Farbpalette ausgewählt. Der Original-Farb-/Grafikadapter von IBM kann nur im Modus 4 (mittlere Auflösung, 4-Farben-Grafik) unter zwei Paletten wählen. Andere Bildschirmadapter haben mehrere Modi, die mit verschiedenen Farbpaletten arbeiten können. Wir wollen hier aber nur die zwei Standardpaletten des Modus 4 besprechen. Die Nummer der Palette wird in BL angegeben; die Palette 0 hat folgendes Aussehen:

0: Momentane Hintergrundfarbe
1: Grün (2)
2: Rot (4)
3: Braun (6)

Palette 1 präsentiert sich als:

0: Momentane Hintergrundfarbe
1: Türkis (3)
2: Magenta (5)
3: Weiß (7)

Im Kapitel 4.3.2.3 finden Sie mehr über die Farbpaletten.

Routinennummer	Parameter
AH = 11	BH = Farbpalettenwert (0 oder 1 für 320 X 200-Grafiken) BL = Farb- oder Farbpalettenwert

Tabelle 9-12 Die von Routine 11 benutzten Register zum Setzen der Farbpalette

9.3.2 Routine 12 (hex C): Pixel setzen

Die Routine 12 erzeugt einen einzigen Pixel. Während sich die Cursorposition, die von den Routinen 9, 10 und 14 verwendet wird, nur auf Zeichen bezieht, erfordert die Routine 12 eine feinere Unterteilung in Rasterzeilen und Rasterspalten. Das Koordinatensystem hat seinen Ursprung mit (0,0) in der linken oberen Ecke des Bildschirmes.
Die Zeilennummer, die nur ein Byte erfordert, wird in Register DL, die Spaltennummer, die zwei Bytes benötigt, in Register CX spezifiziert. Die Farbe steht in AL (direkte Farbe oder *Exklusiv-Oder*-Verknüpfung, wie unter Routine 9 erklärt).
Mehr über Pixel in den Grafikmodi finden Sie in Kapitel 4.4.3.2.

Routinennummer	Parameter
AH = 12	AL = Farbcode des Pixels (0–15) DL = Zeilennummer des Pixels CX = Spaltennummer des Pixels

Tabelle 9-13 Die von Routine 12 verwendeten Register zum Schreiben eines Pixel

9.3.3 Routine 13 (hex D): Pixel abfragen

Die Routine 13 ist die Umkehrung zu Routine 12, sie liest den Inhalt eines Pixels. Jedes Pixel hat nur ein Farbattribut, das von Routine 13 festgestellt wird. Die Routine 8 zum Lesen eines Zeichens liefert sowohl die Farbe als auch einen ASCII-Zeichencode. Die Zeile des Pixels wird in Register DL spezifiziert, nicht in DX (lesen Sie hierzu auch die Anmerkung bei Routine 12), die Spalte in CX. Der Farbcode des Pixels wird in Register AL abgelegt. Alle höherwertigen Bits werden auf Null gesetzt, wie Sie vielleicht schon vermutet haben.

Routinennummer	Parameter
AH = 13	AL = Farbcode des Pixels (0–15) DL = Zeilennummer des Pixels CX = Spaltennummer des Pixels

Tabelle 9-14 Die von Routine 13 verwendeten Register, um die Farbe eines Pixel zu lesen

9.3.4 Routine 14 (hex E): TTY-Zeichen schreiben

Die Routine 14 schreibt einzelne Zeichen auf den Bildschirm, was als *Terminal-* oder *Fernschreiber-* oder TTY-Modus (TTY = *Teletype*) bezeichnet wird. Der Bildschirm verhält sich hierbei wie ein einfacher Drucker, der genau so viel kann, wie für eine Textausgabe nötig ist. Feinheiten wie Farbe, blinkende Zeichen oder Cursorsteuerung sind hier nicht zu finden.
Bei Benutzung der Routine wird das Zeichen an die momentane Cursorposition gesetzt und der Cursor anschließend eine Position nach vorne bewegt, wobei er am Zeilenende an den Anfang der nächsten Zeile springt oder, falls nötig, den Bildschirm rollt. Das zu schreibende Zeichen steht in AL.
In den Textmodi bleiben die Bildschirmattribute erhalten, in den Grafikmodi muß hingegen die Vordergrundfarbe für jedes Pixel neu in Register BL spezifiziert werden.
Es gibt vier ASCII-Zeichen, die die Routine 14 entsprechend ihrer Bedeutung verarbeitet: CHR$(7) - *Beep*, CHR$(8) - *Rücktaste*, CHR$(10) - *Zeilenvorschub* (*Linefeed*), CHR$(13) - *Wagenrücklauf mit Zeilenvorschub* (*Carriage Return*). Alle anderen Zeichen werden normal dargestellt.
Der herausragende Vorteil der Routine ist, daß der Cursor sich mit den geschriebenen Zeichen bewegt, was bei Routine 9 nicht der Fall ist. Mit diesem Hilfsprogramm können hingegen die Farbattribute kontrolliert werden. Wenn wir die beiden Routinen verknüpfen könnten...

Routinennummer	Parameter
AH = 14	AL = zu schreibendes ASCII-Zeichen BL = Vordergrundfarbe des Zeichens (nur in den Grafikmodi) BH = Aktive Anzeigeseite (wird in den Grafikmodi nicht verwendet)

Tabelle 9-15 Die Register, die von Routine 14 zum Schreiben eines TTY-Zeichens verwendet werden

9.3.5 Routine 15 (hex F): Bildschirmmodus feststellen

Die Routine 15 meldet den Bildschirmmodus, der gerade aktiv ist und gibt zwei weitere sehr nützliche Informationen: die Breite des Bildschirmes in Zeichen (80, 40 oder 20) und die angezeigte Seite.
Der Bildschirmmodus wird, wie bei Routine 0 erklärt, im AL-Register abgelegt (die niedrigauflösenden Grafikmodi werden korrekt als 20 Zeichen breit gemeldet). Die angezeigte Seite findet man in BH, sie muß in den Grafikmodi auf 0 gesetzt sein.
Näheres über die Bildschirmmodi finden Sie in Kapitel 4.2. Wie der Modus abgespeichert wird, erfahren Sie in Kapitel 3.2.2, lesen Sie dort bitte unter Speicherstelle hex 449 nach.

9.3.6 Routine 19 (hex 13): Zeichenkette (String) schreiben

Die Routine 19 erlaubt es, eine Zeichenkette (*String*) auf den Bildschirm zu schreiben, leider ist dieses Hilfsprogramm nur im PC AT vorhanden. Durch die vier Unterroutinen, die hier zur Verfügung stehen, läßt sich der Cursor mit dem String bewegen, er kann aber auch an der vorgegebenen Stelle stehen bleiben. Die Attribute können für jedes Zeichen einzeln oder für den gesamten String festgelegt werden.
Die Nummer der Unterroutine wird in AL, der Zeiger, der auf den String deutet, in ES:BP, die Länge des String in CX, die Startposition des String auf dem Bildschirm in DX und die Nummer der betreffenden Seite in BH spezifiziert.
Die Unterroutinen 0 und 1 schreiben einen String, wobei alle Zeichen die in BL abgelegten Attribute erhalten. Mit Unterroutine 0 wird der Cursor nicht von der in DX spezifizierten Stelle bewegt, mit Unterroutine 1 wird er auf dem Zeichen positioniert, das dem letzten Zeichen des Strings folgt.
Die Unterroutinen 2 und 3 schreiben einen String mit individuellen Attributen für jedes Zeichen auf den Bildschirm. Die Unterroutine 2 bewegt den Cursor nicht von der in DX spezifizierten Stelle, während die Unterroutine 3 den Cursor auf die Position nach dem letzten Zeichen des Strings stellt.

Routinennummer	Parameter
AH = 15	AL = momentan aktiver Modus AH = Anzahl der Zeichen pro Zeile BH = aktive Anzeigeseite (0 in den Grafikmodi)

Tabelle 9-16 Die von Routine 15 verwendeten Register zum Lesen des Bildschirmmodus

9.4 Anmerkungen und Beispiel

Nach der Lektüre dieses Kapitels stellt sich Ihnen vermutlich die Frage, ob der direkte Zugriff auf die BIOS-Bildschirmroutinen sinnvoller ist als der Umweg über DOS-Routinen oder die Befehle einer Programmiersprache. Die Antwort kennen Sie schon: Benutzen Sie die höchste Ebene, die Ihren Ansprüchen gerecht wird!
Die Leistungspalette, die das ROM-BIOS im Bereich *Bildschirmsteuerung* zur Verfügung stellt, ist bemerkenswert. Die DOS-Routinen sind im Vergleich sehr dürftig und unterstützen nur ganz einfache Funktionen (Sie werden dies noch in den Kapiteln 14 bis 18 bemerken). Leider verfügen auch viele Programmiersprachen (z.B. Pascal und C) nur über einge-

schränkte Möglichkeiten zur Bildschirmausgabe. Falls Sie nicht mit einer Programmiersprache wie BASIC, deren Befehlsspektrum zur Bildschirmsteuerung sehr ausgeprägt ist, arbeiten, sollten Sie sich auf die direkte Verwendung des ROM-BIOS besinnen. Für eine Steuerung der Bildschirmausgabe sollten die ROM-BIOS-Routinen benutzt werden.
Der direkte Aufruf der ROM-BIOS-Routinen verlangt im allgemeinen eine Assemblerschnittstelle zwischen Programmiersprache und BIOS. Sie sehen nachfolgend ein Beispiel einer Schnittstelle für Pascal. Das Modul setzt den Modus 1 (40-Spalten-Text in Farbe) und definiert die Hintergrundfarbe als blau.
Hier das Assemblermodul:

```
MODUL     SEGMENT 'CODE'
          PUBLIC  BLAU40
BLAU40    PROC    FAR
          PUSH    BP       ;alten BP sichern
          MOV     BP,SP    ;neuen BP setzen
;Bildschirmmodus setzen
          MOV     AH,0     ;Routine 0, Modus setzen
          MOV     AL,1     ;Modus 1: 40-Spalten-Text in Farbe
          INT     16       ;Bildschirmroutine aufrufen
;Hintergrundfarbe setzen
          MOV     AH,11    ;Routine 11, Farbe setzen
          MOV     BH,0     ;Hintergrund setzen
          MOV     BL,1     ;Farbe 1 = Blau
          INT     16       ;Bildschirmroutine aufrufen
          POP     BP       ;alten BP zurückholen
          RET     0        ;zum aufrufenden Programm zurückgeben
BLAU40    ENDP
MODUL     ENDS
          END
```

Kapitel 10
Disketten- und Plattenroutinen im ROM-BIOS

10.1 Standardlaufwerksroutinen im ROM-BIOS 172
10.1.1 Routine 0: Reset der Diskettenstation 173
10.1.2 Routine 1: Laufwerkstatus feststellen 173
10.1.3 Routine 2: Sektoren lesen 173
10.1.4 Routine 3: Sektoren schreiben 175
10.1.5 Routine 4: Sektoren verifizieren 175
10.1.6 Routine 5: Spur formatieren 176
10.1.7 Verwendung der Routine 5 als Kopierschutz 177
10.2 AT-Routinen 178
10.2.1 Routine 8: Aktuelle Laufwerksparameter feststellen 178
10.2.2 Routine 9: Festplattenparametertabelle initialisieren 178
10.2.3 Routine 10 und 11 (hex A und B): Lange Sektoren lesen und schreiben 178
10.2.4 Routine 12: Zylinder anfahren 179
10.2.5 Routine 13: Alternativer Laufwerk-Reset 179
10.2.6 Routine 16: Test, ob Laufwerk bereit 179
10.2.7 Routine 17: Laufwerk neu kalibrieren 179
10.2.8 Routine 20: Controller-Diagnose 179
10.2.9 Routine 21: Laufwerkstyp feststellen 179
10.2.10 Routine 22: Diskettenwechsel erkennen 180
10.2.11 Routine 23: Diskettenlaufwerk festlegen 180
10.3 Laufwerksparametertabelle 180
10.4 Anmerkungen und Beispiele 183

In diesem Kapitel werden die Laufwerksroutinen, die das ROM-BIOS bereitstellt, ausführlich beschrieben. Um den logischen Aufbau einer Diskette zu verstehen, lesen Sie bitte in Kapitel 5 nach. Informationen über Laufwerksroutinen auf höherer Ebene (DOS) finden Sie in Kapitel 15.
Um es gleich zu Anfang zu sagen: Die Diskettenoperationen bleiben am besten dem Diskettenbetriebssystem vorbehalten. Sollten Sie dennoch eine der nachfolgend beschriebenen Routinen direkt benutzen wollen, lesen Sie bitte zuerst den Abschnitt *Anmerkungen und Beispiele* am Ende des Kapitels.

10.1 Standardlaufwerksroutinen im ROM-BIOS

Da Laufwerke nur einige wenige Funktionen ausüben können, gibt es auch nur sechs Diskettenroutinen, die in allen PC-Modellen zu finden sind. Der AT hingegen verfügt über mehrere neue Routinen, da er ein komplexeres Laufwerk besitzt.
Alle Laufwerksroutinen werden durch den Interrupt 19 (hex 13) aufgerufen, wobei die Nummer der BIOS-Routine in das Register AH geladen wird. Wie Sie bestimmt schon erraten konnten, sind die sechs Standardroutinen von 0 bis 5 durchnumeriert.
Alle Diskettenroutinen arbeiten unter Kontrolle der Laufwerksparametertabelle, die Informationen über die Größe der Sektoren, die Zeiten zur Positionierung des Schrittmotors und des Schreib-/Lesekopfes wie auch anderes enthält. Mehr als ein Dutzend Parameter sind in der Laufwerksparametertabelle im ROM abgelegt. Für die meisten Anwendungen ist die Tabelle nicht von Bedeutung. Bei einigen speziellen Anwendungen ist es aber nützlich, den Aufbau der Laufwerksparametertabelle zu kennen. Aus diesem Grund finden Sie am Ende des Kapitels eine kurze Beschreibung.

Routine	Beschreibung
0	Reset der Diskettenstation
1	Diskettenstatus melden
2	Sektoren lesen
3	Sektoren schreiben
4	Sektoren verifizieren
5	Spur formatieren

Tabelle 10-1 Die Standarddiskettenroutinen

10.1.1 Routine 0: Reset der Diskettenstation

Die Routine 0 versetzt den Disketten-Controller und das Laufwerk in den Einschaltzustand. Dieser sog. *Reset* hat keinen Einfluß auf eine Diskette im Laufwerk. Durch das Zurücksetzen wird der Schreib-/Lesekopf auf eine festgelegte Spur gesetzt und die nächstfolgende Diskettenoperation beginnt von dieser Position aus. Programme verwenden die Routine meist, wenn ein Fehler bei einer Diskettenoperation aufgetreten ist.

10.1.2 Routine 1: Laufwerksstatus feststellen

Die Routine 1 schreibt den aktuellen Laufwerksstatus in das Register AL. Der Status jeder Laufwerksoperation wird auf diese Weise festgehalten, unabhängig davon, ob es sich um Lesen, Schreiben, Prüfen (Verifizieren) oder Formatieren handelt. Durch die Speicherung des Status ist es den Routinen für die Fehlerbearbeitung möglich, unabhängig vom Betriebssystem zu arbeiten. Das kann sehr nützlich sein. Unter gewissen Umständen ist es nämlich möglich, DOS oder der Programmiersprache die Steuerung der Diskettenstation zu überlassen (dies sei Ihnen sowieso empfohlen), während gleichzeitig ein Programm abläuft, das Fehler aufdeckt.

Bit 7	6	5	4	3	2	1	0	Wert (Dez)	Bedeutung
1	·	·	·	·	·	·	·	128	Time out: Laufwerk reagiert nicht
·	1	·	·	·	·	·	·	64	Anfahren der gewünschten Spur ist nicht möglich
·	·	1	·	·	·	·	·	32	Controller defekt
·	·	·	1	·	·	·	·	16	Prüfsummenfehler bei CRC-Prüfung
·	·	·	·	1	·	·	·	8	DMA-Fehler
·	·	·	·	·	1	·	·	4	Gefragter Sektor ist auf der Diskette nicht auffindbar (nicht magnetisierbar)
·	·	·	·	·	·	1	·	2	Sektorkennung ungültig oder nicht auffindbar
·	·	·	·	·	·	·	1	1	Ungültiger Befehl an Disketten-Controller gesendet
·	·	·	·	1	·	·	1	9	Überschreitung der DMA-Grenze von 64 Kbyte
·	·	·	·	·	·	1	1	3	Schreibschutzfehler: Versuch, auf geschützter Diskette zu schreiben

Tabelle 10-2 Die Kodierung des Status-Byte, das in Register AL abgelegt wird

10.1.3 Routine 2: Sektoren lesen

Die Routine 2 liest einen oder mehrere Sektoren von der Diskette. Sollen mehrere Sektoren gelesen werden, müssen alle auf einer Seite und auf einer Spur sein, da das ROM-BIOS nicht überprüfen kann, wieviele Sekto-

ren auf einer Spur bzw. Seite untergebracht sind und daher nicht im richtigen Augenblick die Spur (Seite) wechseln kann. Im allgemeinen wird die Routine zum Lesen einzelner Sektoren oder einer ganzen Spur von Sektoren bei DOS-Befehlen wie DISKCOPY und ähnlichen verwendet. Folgende Parameter benötigt die Routine:
DL enthält die Nummer des Laufwerkes.
DH enthält die angesprochene Diskettenseite oder die Nummer des Schreib-/Lesekopfes (0 oder 1).
CH enthält die Nummer der Spur. Diese liegt normalerweise zwischen 0 und 39 (hex 00 bis 27), kann aber auch höher sein - und ist es auch für manche Kopierschutzverfahren. Viele Laufwerke arbeiten bei Formaten mit bis zu 42 Spuren noch korrekt.
CL enthält die Anzahl der Sektoren pro Spur. Der Wert bewegt sich von 1 bis 8 oder 9, kann aber auch (für Kopierschutzzwecke) höher sein. Beachten Sie, daß die Numerierung der Sektoren bei 1 und nicht bei 0 (wie bei Laufwerken, Spuren und Köpfen bzw. Seiten) beginnt.
AL enthält die Anzahl der zu lesenden Sektoren, meist sind dies entweder 1, 8 oder 9. IBM warnt vor der Eingabe einer 0.
ES:BX enthält die Position des Puffers. Der Speicherbereich, in den die von Diskette geholten Daten geschrieben werden, ist durch eine segmentierte Adresse im ES:BS-Registerpaar bezeichnet. Das Registerpaar ES:BS dient den meisten BIOS-Routinen zur Ablage segmentierter Adressen.
Der Speicherbereich sollte groß genug sein, um alle gelesenen Daten aufnehmen zu können. Beachten Sie, daß normale DOS-Sektoren zwar nur eine Länge von 512 Bytes aufweisen, generell Sektoren aber bis zu 1.024 Bytes lang sein können (sehen Sie sich hierzu auch die noch folgende Formatroutinen an). Mehrere gelesene Sektoren werden nacheinander im Speicher abgelegt.
CF enthält den Fehlerstatus der Operation. Das Resultat wird durch eine Kombination der Carry-Flagge (CF) mit dem AH-Register ausgedrückt. Ist CF gleich 0 trat kein Fehler auf und AH ist folglich ebenfalls 0. Bei CF gleich 1 war ein Fehler zu verzeichnen und AH enthält die Statusbits, die schon bei Routine 1 (Diskettenstatus melden) erläutert wurden.
Bei der Benutzung der Routine 2 oder einer anderen Routine, die aktiv auf das Laufwerk zugreift, müssen Sie beachten, daß der Laufwerksmotor eine gewisse Zeit braucht, um seine Arbeitsgeschwindigkeit zu erreichen. Keine der Routinen wartet mit der Ausführung, bis der Motor die richtige Geschwindigkeit erreicht hat. Das führt allerdings selten zu Problemen, wenn man folgenden Ratschlag der IBM befolgt: Ein Programm soll eine Routine dreimal aufrufen, bevor eine Fehlermeldung erfolgt. Zwischen jedem Versuch soll ein Reset durchgeführt werden. Das Prinzip können Sie aus folgendem Beispiel ersehen, das teilweise in BASIC gehalten ist:

```
10 FEHLERZAHL = 0
20 WHILE FEHLERZAHL < 3
30 'Lese-, Schreib-, Verifizier- oder Formatier-Befehl
40 'Fehlerabfrage: falls kein Fehler: weiter bei 90
50   FEHLERZAHL = FEHLERZAHL + 1
60 'Reset wird ausgeführt
70 WEND
80 'Auf Fehler reagieren
90 'Wenn erfolgreich, im Programm fortfahren
```

Die Auswirkungen der Reset-Operation auf die Laufwerksparametertabelle ist in Kapitel 10.3 nachzulesen.

Parameter	Resultierender Status
DL = Laufwerksnummer	Falls CF = 0, wird AH auch 0; kein Fehler
DH = Nummer der Diskettenseiten oder des Schreib-/Lesekopfes	Falls CF = 1, enthält AH die Statusbits für Routine 1; Fehler aufgetreten
CH = Spurnummer	
CL = Sektornummer	
AL = Anzahl der zu lesenden Sektoren	
ES:BX = Anfang des Pufferspeichers	

Tabelle 10-3 Die Register für die Kontrollparameter der Routinen Lesen, Schreiben, Verifizieren und Formatieren

10.1.4 Routine 3: Sektoren schreiben

Die Routine 3 schreibt in Sektoren auf eine Diskette, sie ist das Gegenstück zu Routine 2. Alle Erläuterungen, die Sie zu Routine 2 finden, gelten auch für dieses Hilfsprogramm. Eine Diskette muß formatiert sein, bevor in einen Sektor geschrieben werden kann.

10.1.5 Routine 4: Sektoren verifizieren

Routine 4 prüft den Inhalt einer oder mehrerer Sektoren einer Diskette. Viele PC-Benutzer machen sich von diesem Prüfprozeß eine falsche Vorstellung. Es werden nicht die Daten auf der Diskette mit denen im Hauptspeicher des Computers verglichen, sondern es wird lediglich geprüft, ob alle Sektoren der Diskette lesbar sind und eine korrekte Prüfsumme aufweisen. Der CRC-Prüfsummentest (*Cyclical Redundancy Check*) ist ein Paritätstest für die Daten jedes Sektors, der sehr zuverlässig die meisten Fehler findet.

Routine 3 wird häufig nach einem schreibenden Diskettenzugriff aufgerufen, um die Sicherheit des Schreibvorganges zu erhöhen. Prinzipiell kann die Routine jeden Diskettenbereich überprüfen. Da die Diskettenlaufwerke heute relativ zuverlässig arbeiten und die normalen Fehlermeldungen sehr gut funktionieren, erachten viele Programmierer die Routine 3 als überflüssig. DOS ruft sie noch nicht einmal nach der Operation *Sektoren schreiben* auf, es sei denn, dies wird mit VERIFY ON spezifiziert.
Die Routine arbeitet exakt wie die zwei zuvor besprochenen (Lesen und Schreiben von Sektoren) mit der Ausnahme, daß sie keinen Speicherbereich benötigt und daher das Register-Paar ES:BX nicht verwendet.

10.1.6 Routine 5: Spur formatieren

Die Routine 5 formatiert eine Spur auf einer Seite der Diskette. Sie ist den Routinen 2 und 3 sehr ähnlich aufgebaut. Die Sektornummer im Register CL wird nicht benutzt, alle anderen Parameter werden wie bei den Routinen 2 und 3 erklärt verwendet.
Formatiert wird stets die ganze Spur, die Formatierung eines einzelnen Sektors ist nicht möglich. Trotzdem können wir für die einzelnen Sektoren individuelle Merkmale spezifizieren.
Jeder Sektor einer Spur hat vier Bytes, die im Datenbereich stehen und den Sektor beschreiben. Diese Bytes zur Sektorerkennung werden für Prozesse wie Lesen, Schreiben und Verifizieren verwendet. Man spricht von den vier Adreßbytes als C für *Zylinder* (*Cylinder*), dem allgemeinen Ausdruck für *Spur*, H für *Kopf* (*Head*) oder *Seite*, R für *Eintrag* (*Record*) oder *Sektornummer* und N für die Anzahl der *Bytes pro Sektor* (*Number*), was auch als *Längencode* (*Size Code*) bezeichnet wird.
Wenn ein Sektor gelesen oder geschrieben wird, sucht das ROM-BIOS die Diskettenspur nach den vier Sektorbytes ab, von denen R, die Sektornummer, am wichtigsten ist. Zylinder- und Kopfparameter werden nicht unbedingt benötigt, da die Spur auf mechanischem und die Seite auf elektronischem Wege korrekt ausgewählt wird. Die Sektorbytes C und H werden dennoch aufgezeichnet, um eine doppelte Sicherheit zu haben.
Der Längencode (N) kann einen von vier Standardwerten annehmen; der normale Wert ist 2, das entspricht 512 Bytes.

N	Sektorlänge (Byte)	Sektorlänge (Kbyte)
0	128	1/8
1	256	1/4
2	512	1/2
3	1024	1

Tabelle 10-4 Die vier Standardwerte des Längencode N

Die Sektoren werden in der Reihenfolge auf die Diskette geschrieben, die sich aus den Adreßbytes ergibt. Dies muß keine sequentielle Reihenfolge zu sein. Die Anordnung der Sektoren kann so gewählt werden, daß beispielsweise eine kürzere Zugriffszeit oder ein Kopierschutz erreicht wird. Auf der Festplatte des XT sind logisch aufeinanderfolgende Sektoren um je sechs Sektoren verschoben. Auf DOS-Disketten hingegen sind Programme sowohl logisch als auch physikalisch sequentiell, also: 0, 1, 2, usw., abgespeichert.
Für die normale 9-Sektoren-Diskette des DOS sieht die Formatadressierung der Spur 0 auf Seite 1 folgendermaßen aus:

```
CHRN  CHRN  CHRN  ...  CHRN
0112  0122  0132  ...  0192
```

Wenn eine Spur formatiert wird, orientiert sich das Laufwerk am Indexloch und benutzt es als Startmarkierung. Bei allen anderen Operationen ist das Indexloch ohne Bedeutung.
Beachten Sie bitte, daß die Formatierungsroutine keinen Wert festlegt, der in die Sektoren geschrieben wird. Die Verwaltung der Sektoreninhalte ist die alleinige Aufgabe der Laufwerksparametertabelle. Informationen über diese Tabelle finden Sie in Kapitel 10.3.

10.1.7 Verwendung der Routine 5 als Kopierschutz

Spuren lassen sich auf auf jedes beliebige Format formatieren, die meisten Betriebssysteme können allerdings nur bestimmte Formate lesen. Die Konsequenz ist, daß viele Kopierschutzverfahren auf exotischen Formaten basieren, die den Betriebssystemen das Lesen und Kopieren der abgespeicherten Daten unmöglich machen. Wir haben verschiedene Methoden zu unterscheiden:

Wir können die Ordnung der Sektoren umstellen, wodurch die Zugriffszeit in einer Weise verändert wird, die das Kopierschutzverfahren erkennt.

Wir können mehr Sektoren auf einer Spur unterbringen (10 ist die maximale Anzahl an Sektoren bei einer Länge von 512 Bytes).

Wir können eine Sektornummer weglassen.

Wir können Sektoren mit ungewöhnlichen Sektornummern hinzufügen (z.B. R gleich 22).

Wir können einem oder mehreren Sektoren eine ungewöhnliche Länge geben.

Wir können falsche Werte für C und H ablegen.

Hinweis: Die Wirksamkeit eines Kopierschutzes hängt nicht davon ab, ob er von DOS erkannt wird.

10.2 AT-Routinen

Der AT verwendet Laufwerke, die sich erheblich von denen der anderen PC-Modelle unterscheiden. Aus diesem Grund wurden einige neue Diskettenroutinen beim AT hinzugefügt. Diese Laufwerke unterstützen Disketten mit höherer Speicherkapazität und eine Vielzahl von Festplatten, die der AT benutzen kann. Wir erwähnen diese Routinen hier, beschreiben sie aber nicht im Detail, da sie eine Eigenart des AT sind und nicht der in diesem Buch vorgesehenen Zielrichtung folgen, die Programmierpraktiken zu besprechen, die auf die gesamte PC-Familie zutreffen.

10.2.1 Routine 8: Aktuelle Laufwerksparameter feststellen

Routine 8 zeigt die aktuellen Laufwerksparameter an. DL enthält die Anzahl der angeschlossenen Diskettenlaufwerke (0 bis 2), DH die maximale Kopf/Seiten-Zahl, CH das maximale Verhältnis Zylinder pro Spur und CL die höchste Sektornummer.

10.2.2 Routine 9: Festplattenparametertabelle initialisieren

Die Routine 9 stellt die Laufwerksparametertabelle (in diesem Zusammenhang auch *Festplattenparameterliste*) auf zwei Festplattenlaufwerke ein. Die Interruptvektoren der Interrupts 65 (hex 41) und 70 (hex 46) werden verwendet, um die Tabellenadressen bereitzustellen. Diese Routine wird nur dann eingesetzt, wenn ein "fremdes" Laufwerk angeschlossen ist.

10.2.3 Routine 10 und 11 (hex A und B): Lange Sektoren lesen und schreiben

Routine 10 liest und Routine 11 schreibt lange ECC-Sektoren auf einer 20 Mbyte Festplatte. Ein langer Sektor enthält einen Fehlercode aus vier Bytes, den ECC, der zur Fehlererkennung und korrektur verwendet wird.

10.2.4 Routine 12 (hex C): Zylinder anfahren

Die Routine 12 führt eine Suchoperation durch, die den Schreib-/Lesekopf über einem bestimmten Zylinder einer Festplatte positioniert. Register DL enthält die Laufwerksnummer, DH die Nummer des Kopfes und CH die Nummer des Zylinders.

10.2.5 Routine 13 (hex D): Alternativer Laufwerks-Reset

Die Routine 13 übernimmt einen Laufwerks-Reset für die Festplattenlaufwerke. Das Laufwerk wird in Register DL spezifiziert. Das Hilfsprogramm arbeitet in gleicher Weise wie Routine 0.

10.2.6 Routine 16 (hex 10): Test, ob Laufwerk bereit

Die Routine testet, ob das Laufwerk bereit ist. Das Laufwerk ist in Register DL spezifiziert, der Status in AH.

10.2.7 Routine 17 (hex 11): Laufwerk neu kalibrieren

Routine 17 kalibriert Festplattenlaufwerke neu. Das Laufwerk wird in Register DL spezifiziert und das Ergebnis, der Status, in Register AH geschrieben.

10.2.8 Routine 20 (hex 14): Controller-Diagnose

Die Routine 20 ruft eine interne Diagnoseroutine des Disketten-Controllers des AT auf. Der Status des Controllers steht in Register AH.

10.2.9 Routine 21 (hex 15): Laufwerkstyp feststellen

Die Routine wurde eingebaut, um jederzeit den Typ des verwendeten Laufwerkes bestimmen zu können. Die Laufwerksnummer wird in DL spezifiziert. Routine 21 meldet bei Aufruf im Register AH, welcher Laufwerkstyp unter der Nummer angeschlossen ist: AH gleich 0 bedeutet, daß kein Laufwerk angeschlossen ist. Steht in AH der Wert 1, handelt es sich um ein Laufwerke, das den Diskettenwechsel nicht bemerkt. Das ist bei den meisten Laufwerken der Fall. AH gleich 2 zeigt ein Laufwerk an, das einen Diskettenwechsel erkennt, das trifft z.B. auf die Diskettenlaufwerke mit 1,2 MByte des AT zu. Ist ein Festplattenlaufwerk angeschlos-

sen, erhält das Register AH den Wert 3 zugewiesen. In letzterem Fall wirkt das Registerpaar CX:DX als 4-byte-Ganzzahl, die die Anzahl der Sektoren in diesem Laufwerk angibt.

10.2.10 Routine 22 (hex 16): Diskettenwechsel erkennen

Routine 22 meldet einen Wechsel der Diskette in Laufwerken, die dieses erkennen können, wie z.B. die Laufwerke des AT. Steht das Register AH auf 0, hat kein Wechsel stattgefunden. Wird nun gewechselt, ändert sich AH auf 6, während in Register DL die Nummer des Laufwerkes abgespeichert wird, in dem der Wechsel stattgefunden hat.
Die Möglichkeit, einen Diskettenwechsel zu erkennen, kann sehr nützlich sein. Für bestimmte kritische Diskettenoperationen wie dem Lesen einer Dateizuordnungstabelle (FAT) ist es sehr wichtig zu wissen, ob die Diskette gewechselt wurde oder nicht. Wurde ein Wechsel durchgeführt, müssen die Diskettendaten im Computer gelöscht und neue Daten gelesen werden.

10.2.11 Routine 23 (hex 17): Diskettenlaufwerk festlegen

Die Routine 23 wird verwendet, um die Art eines angeschlossenen Diskettenlaufwerkes festzulegen. AL gleich 0 bedeutet, daß kein Laufwerk angeschlossen ist, AL gleich 1 zeigt eine normale Diskette mit normalem Laufwerk an, AL gleich 3 steht für Disketten mit 1,2 Mbyte mit dem entsprechenden Laufwerk. Diese Routine wird zusammen mit der Formatierungsroutine (Routine 5) verwendet, um den Diskettentyp, der formatiert werden soll, zu bestimmen.

10.3 Laufwerksparametertabelle

Alle Diskettenoperationen werden von einem Parameterblock gesteuert, den man *Laufwerksparametertabelle* nennt. Es gibt eine Version dieser Tabelle im ROM, an der standardisierten Speicherstelle F000:EFC7, wir können aber auch eine eigene Tabelle erstellen. Die neue Tabelle kann die alte ersetzen, indem sie in den Speicher geladen wird und der Vektor des Interrupts 30 (hex 1E) auf die neue Tabelle gerichtet wird. Jede DOS-Version verwendet eine eigene Tabelle, statt der im ROM bereitgestellten.
Die Laufwerksparametertabelle setzt sich aus elf Bytes zusammen. Wir gehen die Tabelle Byte für Byte durch und vergleichen die von der Version 2.10 verwendeten Werte mit den im ROM vorhandenen. Die meisten dort abgelegten Informationen sind für Programmierer von geringem Nutzen, es sei denn, man will eine neue Tabelle schreiben und die vom DOS verwendete ersetzen.

Byte Offset	Verwendung
0	Spezifiziert Byte 1: Schrittzeit des Motors, Kopfschreibzeit
1	Spezifiziert Byte 2: Kopfladezeit, DMA-Modus
2	Wartezeit, bis Motor ausgeschaltet ist
3	Bytes pro Sektor: 0 = 128; 1 = 256; 2 = 512; 3 = 1024
4	Letzte Sektornummer
5	Freiraum zwischen Sektoren für Formatierungsoperationen
6	Datenlänge, falls Sektorenlänge nicht spezifiziert
7	Freiraum zwischen Sektoren für Formatierungsoperatoren
8	Datenwert in formatierten Sektoren
9	Kopfruhezeit zum Abklingen der Vibration
A	Anlaufzeit des Motors

Tabelle 10-5 Die elf Bytes in der Laufwerksparametertabelle

Die Bytes 0 und 1 sind Teil des Befehlsstrings, der an den Floppydisk-Controller (FDC) übermittelt wird. Die ersten vier Bit des ersten Bytes enthalten die Schrittzeit (*Step-Rate Time* oder SRT), das ist die Zeit, die das ROM-BIOS dem Laufwerk zur Verfügung stellt, damit der Schreib-/Lesekopf von Spur zu Spur springen kann. Der Wert im ROM liegt bei 8 Millisekunden, DOS 2.10 verkürzt diese Zeit auf 6 Millisekunden, wodurch die Diskettenoperationen schneller ablaufen. Im ersten Byte wird auch der DMA-Modus gesetzt.

Byte 2, Offset-Adresse 2, spezifiziert, wie lange der Motor des Laufwerkes in Betrieb bleibt, nachdem eine Diskettenoperation abgeschlossen wurde. Der Motor wird nicht sofort abgeschaltet, da ein weiterer Zugriff unmittelbar folgen könnte. Der Wert wird in Systemtakteinheiten (ca. 18 Einheiten pro Sekunde) bestimmt. Alle Versionen der Tabelle haben hier den Wert 37 (hex 25), was einer Nachlaufzeit von ungefähr 2 Sekunden entspricht.

Byte 3, Offset-Adresse 3, gibt den Längencode der Sektoren an; das ist derselbe Code N, der bei der Formatierungsroutine (Routine 5) verwendet wird. Für gewöhnlich enthält Byte 3 den Wert 2, der die übliche Länge von 512 Byte pro Sektor spezifiziert. Für jede Lese , Schreib- und Verifizieroperation muß der Code auf den richtigen Wert gesetzt werden, dies gilt insbesondere dann, wenn wir es mit Sektoren ungewöhnlicher Länge zu tun haben.

Byte 4, Offset-Adresse 4, gibt die Nummer des letzten Sektors einer Spur an. Für DOS 2.10 ist dieser Wert 9, der im ROM abgelegte Wert beträgt 8.

Byte 5, Offset-Adresse 5, spezifiziert die Länge des Freiraumes zwischen den Sektoren, der zu berücksichtigen ist, wenn Daten gelesen oder geschrieben werden. Es zeigt dem ROM-BIOS an, wie lange es warten muß,

bevor der nächsten Sektors erreicht wird. Auf diese Weise wird das Schreiben oder Lesen unsinniger Stellen auf der Diskette verhindert. In der Standardtabelle liegt der Wert bei 42 (hex 2A).
Byte 6, Offset-Adresse 6, wird *Datentransferlänge* (*Data Transfer Length* oder DTL) genannt und hat einen Wert von 255 (hex FF). Das Byte bestimmt die maximale Datenlänge, wenn die Sektorlänge nicht spezifiziert ist.
Byte 7, Offset-Adresse 7, spezifiziert die Länge der magnetischen Lücke zwischen Sektoren, wenn eine Spur formatiert wird. Diese Lücke ist natürlich größer als der Suchfreiraum, der in Byte 5 festgelegt wird. Der Standardwert liegt bei 80 (hex 50).
Byte 8, Offset-Adresse 8, stellt den Datenwert bereit, der in jedem Byte eines formatierten Sektors gespeichert wird. Der Standardwert ist hex F6, das Divisionssymbol. Sie können jedes andere Zeichen in Byte 8 speichern.
Byte 9, Offset-Adresse 9, setzt die Kopfruhezeit fest. Das ist die Zeit, die das Laufwerk dem Schreib-/Lesekopf nach dem Anfahren einer neuen Spur gewährt, damit die Vibrationen abgeklingen können. Die im ROM gespeicherte Tabelle sieht 25 (hex 19) Millisekunden vor, DOS hingegen nur 15 (hex F) Millisekunden.
Byte 10, das letzte Byte der Tabelle an der Offset-Adresse 10, bestimmt die Zeit, die dem Laufwerksmotor bleibt, um auf seine Betriebsgeschwindigkeit zu kommen. Sie wird in 1/8 Sekunden gemessen. Im ROM finden wir den Wert 4, was einer halben Sekunde entspricht. DOS verkürzt auch hier die Zeit, und zwar mit dem Wert 2 um die Hälfte auf 1/4 Sekunde.
Da in der Laufwerksparametertabelle genügend Parameter stehen, die wir verändern können, steht uns bei Experimenten eine Vielzahl von "Aufregungen" und "Gefahren" bevor, die wir hier nicht erfassen können. Das folgende Beispiel zeigt Ihnen, wie Sie den Wert, der beim Formatieren in jedes Byte geschrieben wird, ändern können. Wir stellen den Wert von hex F6 auf hex AA um, nur um zu zeigen, wie dies zu bewerkstelligen ist:

```
10 DEF SEG
20 OFFSET = PEEK (120 + 0) + 256 + * PEEK (120 + 1)
                                        'Offset des Diskettenbasisvektors
30 SEGMENT = PEEK (120 + 2) + 256 * PEEK (120 + 3)
                                        'Segment des Diskettenbasisvektors
40 DEF SEG = SEGMENT
45 'Liegt das Segment im oberen Speicherbereich, handelt es sich
   'um ROM und kann nicht geändert werden
50 IF SEGMENT >= &HF000 THEN PRINT "ROM-Diskettentabelle"
60 FORMAT.DATEN =  PEEK (OFFSET + 8)     'alten Daten auslesen
70 PRINT "Format setzt Daten auf" HEX$(FORMAT.DATEN)
80 POKE OFFSET + 8, &HAA      'Datenwert hex AA schreiben
```

10.4 Anmerkungen und Beispiele

Während der direkte Zugriff auf die im letzten Kapitel behandelten BIOS-Bildschirmroutinen gefahrlos ist, sollten die in diesem Kapitel besprochenen BIOS-Routinen aus Sicherheitsgründen nur in Ausnahmefällen benutzt werden. Die normalen Diskettenoperationen sollten Sie DOS (siehe Kapitel 14 bis 18) oder einer Programmiersprache überlassen. Meistens ist es nicht nötig, tiefer in die Materie einzudringen. Es gibt aber Situationen, in denen es nicht zu vermeiden ist, z.B. zum Schutz gegen das Kopieren. Das ist einer der Fälle, in denen wir direkt auf das ROM-BIOS zugreifen müssen.
Als Beispiel finden Sie zum Abschluß des Kapitels einige Unterprogramme in Pascal, die Diskettensektoren lesen und schreiben können. Wir beginnen mit der Pascal-Seite der Schnittstelle. Das folgende Programm zeigt diesen Aspekt. Sollten Sie im Umgang mit Pascal nicht geübt sein und wollen die Routine auch nicht mühsam entziffern, überspringen Sie sie und studieren das danach folgende Beispiel in Assembler.

```
PROGRAM DISKETTEN_SCHNITTSTELLE;
{definiert unseren Schreib-/Lese-Bereich}
  TYPE
    SEGMENT_TYP = ARRAY [0..511] OF BYTE;
  VAR
    SEGMENT_DATEN : SEGMENT_TYP;
{definiert die Assembler-Schreib-/Leseroutinen}
  FUNCTION  SEGLESEN (
                  VAR   S : SEGMENT_TYP;   {Datenbereich}
                        D : INTEGER;       {Laufwerksnummer}
                        C : INTEGER;       {Nummer der Spur}
                        H : INTEGER;       {Nummer des Kopfes}
                        R : INTEGER )      {Segmentnummer}
            :   BYTE;                      {Statuscode-Byte}

    EXTERNAL;
FUNCTION  SEGSCHREIBEN (
                   VAR   S : SEGMENT_TYP    {Datenbereich}
                         D : INTEGER;       {Laufwerksnummer}
                         C : INTEGER;       {Nummer der Spur}
                         H : INTEGER;       {Nummer des Kopfes}
                         R : INTEGER )      {Segmentnummer}
            :   BYTE;                       {Statuscode-Byte}
    EXTERNAL;
{hier ein kurzes Programm, das das Urladersegment liest:}
  BEGIN
    IF SEGLESEN ( SEGMENT_DATEN, 0, 0, 0, 1 ) = 0
      THEN                              {kein Fehler}
      ELSE ;                            {Fehler}
  END.
```

Durch die zwei Hilfsprogramme ist der Rahmen abgesteckt und wir können zur Schnittstellenroutine in Assembler übergehen.
Es liegen zwei Assembleroutinen vor, die beide bis auf ihre Namen und die Codes der BIOS-Routinen, die sie aufrufen, identisch sind. Der Aufruf der Parameter erfolgt in der Form [BP] + x. Im unserem Fall ist der erste Parameter des Stapels eine Offset-Adresse, die anderen Parameter sind die aktuellen Werte wie Laufwerksnummer usw. Obwohl der erste Parameter eine Adresse und kein Wert ist, wird er genauso wie alle anderen behandelt, da nicht der Inhalt der Speicherstelle interessiert, sondern deren Adresse. Die Adresse wird direkt an die ROM-BIOS-Routine übergeben.
Nachfolgend nun die Schreib-/Lese-Schnittstelle zum Einsatz unter Pascal, die aus zwei nahezu identische Prozeduren zum Lesen bzw. Schreiben von Diskettendaten besteht:

```
SCHNITTSTELLE SEGMENT 'CODE'
              PUBLIC  SEGLESEN
              PUBLIC  SEGSCHREIBEN
;Leseroutine:
SEGLESEN      PROC    FAR
              PUSH    BP
              MOV     BP,SP
              PUSH    DS              ;DS nach ...
              POP     ES              ;... ES transferieren
              MOV     BX,[BP+14]      ;Daten-Offset angeben
              MOV     DL,[BP+12]      ;Laufwerksnummer angeben
              MOV     CH,[BP+10]      ;Nummer der Spur angeben
              MOV     DH,[BP+08]      ;Nummer der Seite angeben
              MOV     CL,[BP+06]      ;Sektornummer angeben
              MOV     AL,1            ;Abfrage nach einem Sektor
              MOV     AH,2            ;Leseroutine anwählen
              INT     19              ;Diskettenroutine aufrufen
              MOV     AL,AH           ;Status ablegen
              POP     BP
              RET     10              ;10 Parameter auf dem Stapel
SEGLESEN      ENDP
```

```
;Schreibroutine (bis auf die Routinennummer identisch)
SEGSCHREIBEN  PROC    FAR
              PUSH    BP
              MOV     BP,SP
              PUSH    DS            ;DS nach ...
              POP     ES            ;... ES transferieren
              MOV     BX,[BP+14]    ;Daten-Offset angeben
              MOV     DL,[BP+12]    ;Laufwerksnummer angeben
              MOV     CH,[BP+10]    ;Nummer der Spur angeben
              MOV     DH,[BP+08]    ;Nummer der Seite angeben
              MOV     CL,[BP+06]    ;Sektornummer angeben
              MOV     AL,1          ;Abfrage nach einem Sektor
              MOV     AH,3          ;Schreibroutine aufrufen
              INT     19            ;Diskettenroutine aufrufen
              MOV     AL,AH         ;Status ablegen
              POP     BP
              RET     10            ;10 Parameter auf dem Stapel
SEGSCHREIBEN  ENDP
SCHNITTSTELLE ENDS
              END
```

Kapitel 11
Die Tastaturroutinen im ROM-BIOS

11.1 Zugriff auf die Tastaturroutinen 188
11.1.1 Routine 0: Zeichen des nächsten Tastenanschlags holen 188
11.1.2 Routine 1: Meldung, ob Zeichen bereit 188
11.1.3 Routine 2: Umschaltstatus feststellen 189
11.2 Anmerkungen und Beispiel 190

Obwohl es nur wenige Tastaturroutinen des ROM-BIOS gibt und diese auch nicht so kompliziert sind wie die Bildschirm- (Kapitel 9) oder die Floppy/Plattenroutinen (Kapitel 10), sind sie doch wichtig genug, um zu einem eigenen Kapitel zusammengefaßt zu werden. Die restlichen ROM-BIOS-Routinen sind in Kapitel 12 zusammengefaßt.

11.1 Zugriff auf die Tastaturroutinen

Die Tastaturroutinen werden mit Interrupt 22 (hex 16) aufgerufen. Es gibt drei verschiedene Hilfsprogramme, die die Nummern 0 bis 2 tragen. Wie stets im ROM-BIOS werden die Routinen über die Routinennummer im Register AH ausgewählt.

Routine	Beschreibung
0	Zeichen des nächsten Tastenanschlags holen
1	Meldung, ob Zeichen bereit
2	Umschaltstatus feststellen

Tabelle 11-1 Die drei Tastaturroutinen in ROM-BIOS

11.1.1 Routine 0: Zeichen des nächsten Tastenanschlags holen

Routine 0 meldet das Zeichen eines Tastenanschlags. Liegt bereits ein Zeichen im ROM-BIOS-Tastaturpuffer vor, erfolgt die Meldung augenblicklich, andernfalls wartet die Routine, bis ein Zeichen in den Tastaturpuffer geschrieben wird. Wie Sie in Kapitel 6.2 gesehen haben, erfolgt die Übermittlung der einzelnen Zeichen durch ein Byte-Paar, das sich aus Haupt- und Hilfs-Byte zusammensetzt. Das Haupt-Byte, das im AL-Register steht, ist entweder 0 für spezielle Zeichen (z.B. Funktionstasten) oder ein ASCII-Code für normale ASCII-Zeichen. Das Hilfs-Byte, das im AH-Register steht, ist entweder ein Kennzeichen für Sonderzeichen oder der standardisierte Tastaturauswahlcode der PC-Tastatur für ASCII-Zeichen.
Steht im Tastaturpuffer kein Zeichen bereit, wenn Routine 0 aufgerufen wird, wartet sie, bis ein Zeichen eintrifft. Dadurch wird der Programmablauf quasi *eingefroren*. Man kann dies unterbinden, indem man Routine 1 anstelle von Routine 0 wählt.
Im Gegensatz zu den Ausführungen in einigen IBM-Handbüchern sind die Routinen 0 und 1 sowohl für normale ASCII-Zeichen als auch für Sonderzeichen wie beispielsweise die von den Funktionstasten erzeugten Zeichen verwendbar.

11.1.2 Routine 1: Meldung, ob Zeichen bereit

Routine 1 meldet, ob ein Zeichen im Tastaturpuffer zum Lesen bereitsteht. Es wird nur festgestellt, ob ein Zeichen vorliegt oder nicht; das Zeichen bleibt im Puffer, bis es mit Routine 0 gelesen wird. Die Zeroflagge speichert das Ergebnis von Routine 1: Bei ZF gleich 0 steht ein

Zeichen bereit, bei ZF gleich 1 liegt kein Zeichen im Puffer vor. Beachten Sie bitte, daß die Statusmeldung hier verdreht ist: 0 bedeutet *ja*, 1 steht für *nein*. Ist ein Zeichen vorhanden, erhalten die Register AH und AL dieselben Werte zugewiesen wie durch die Routine 0.
Die Routine 1 ist insbesondere für zwei häufig benötigte Programmoperationen gut einsetzbar. Bei der ersten überprüft ein Programm, ob eine Taste betätigt wurde und reagiert darauf bzw. setzt die Programmausführung fort, falls kein Tastendruck vorliegt. Üblicherweise kann man mit dieser Methode eine Programmabarbeitung mit einem Tastendruck anhalten. Die andere Programmoperation ist das Löschen des Tastaturpuffers. Meistens ist es für den Benutzer angenehm, wenn er schneller eintippen kann, als das Programm Tastenanschläge verarbeitet. Bei bestimmten Anwendungen ist diese Methode allerdings nicht angebracht, etwa bei der Abfrage "Wollen Sie Ihre Daten löschen?". Unter diesen Umständen müssen die Programme eine Möglichkeit erhalten, den Tastaturpuffer zu leeren, d.h., alle Eingaben zu löschen. Der übliche Weg führt über die Routinen 0 und 1. Hier sehen Sie ein Programmgerüst zum Löschen des Tastaturpuffers:

```
10 Aufruf Routine 1 und Test, ob ein Zeichen vorliegt
20 WHILE ZF = 0
30 Routine 0 aufrufen, um das Zeichen zu holen
40 Aufruf Routine 1 und Test, ob ein weiteres Zeichen bereitsteht
50 WEND
```

Die Routinen 0 und 1 können beide sowohl für ASCII-Zeichen als auch für Sonderzeichenzeichen, z.B. Funktionstasten, eingesetzt werden.

11.1.3 Routine 2: Umschaltstatus feststellen

Routine 2 setzt das Register AL je nach Umschaltungsstatus der Tastatur. Der Umschaltstatus wird Bit für Bit aus dem ersten Tastaturstatus-Byte entnommen, das in der Speicherstelle hex 417 zu finden ist. Wichtige Informationen über die Tastaturstatus-Bytes (hex 418 und hex 488) finden Sie in Kapitel 3.
Die Routine 2 ist im allgemeinen nicht besonders nützlich, es sei denn, Sie wollen die Tastatur mit einer neuen Tastenbelegung versehen. Die Routine läßt sich z.B. einsetzen, um zwischen dem Drücken der linken und der rechten Umschalttaste (Shifttaste) zu unterscheiden.

Bit 7	6	5	4	3	2	1	0	Bedeutung	
x	·	·	·	·	·	·	·	Einfügemodus:	1 = aktiv
·	x	·	·	·	·	·	·	Caps-Lock-Modus:	1 = aktiv
·	·	x	·	·	·	·	·	Num-Lock-Modus:	1 = aktiv
·	·	·	x	·	·	·	·	Scroll-Lock-Modus:	1 = aktiv
·	·	·	·	x	·	·	·	Alt-Modus:	1 = aktiv (Alt-Taste gedrückt)
·	·	·	·	·	x	·	·	Ctrl-Modus:	1 = aktiv (Ctrl-Taste gedrückt)
·	·	·	·	·	·	x	·	Umschaltmodus:	1 = aktiv (linke Shifttaste gedrückt)
·	·	·	·	·	·	·	x	Umschaltmodus:	1 = aktiv (rechte Shifttaste gedrückt)

Tabelle 11-2 Die Tastaturstatus-Bits im Register AL für Routine 2

11.2 Anmerkungen und Beispiel

Auf die Frage, welche Routinen sich besser zur Tastaturabfrage eignen (DOS oder BIOS), gibt es keine eindeutige Antwort. Da aber DOS generell der Vorzug zu geben ist, da es hardwareunabhängiger als das ROM-BIOS ist, sollten Sie auch die BIOS-Tastaturroutinen nur verwenden, wenn DOS die entsprechenden Funktionen nicht bereitstellt. Das ist selten der Fall, da DOS im Tastaturbereich recht leistungsstark ist.
Die meisten Programmiersprachen greifen auf die DOS-Routinen zurück, um Tastaturoperationen durchzuführen. Der wichtigste Vorteil der DOS- gegenüber den BIOS-Routinen ist, daß DOS eine Stringeingabe unter vollständiger Kontrolle (mit allen Editiertasten) zur Verfügung stellt. Die Eingabe wird mit einem Druck auf die Return-Taste abgeschlossen. Vorausgesetzt, ein Programm muß die Eingaben nicht Zeichen für Zeichen überwachen, sparen Sie sich viel Programmieraufwand (und Probleme mit den Benutzern), wenn Ihr Programm die Eingaben als String von DOS direkt oder indirekt über die Programmiersprache übernimmt. Wenn Sie eine vollständige Kontrolle über die Eingabe benötigen, liegt es nahe, die BIOS-Routinen anstelle der DOS-Routinen zu verwenden.
Das Beispielprogramm in Assembler dieses Kapitels ist etwas phantasievoller als die Beispiele der vorangegangenen Kapitel ausgefallen. Es geht um das Löschen des Tastaturpuffers. Das Programm erledigt die unter Routine 1 nur angedeutete Löschung des Puffers.
In unserem Programm ist der Gebrauch von Labels (Marken) und Verzweigungen für Sie neu. In der Erläuterung der Grundregeln der Assemblerschnittstellen in Kapitel 8, wurde eine Anweisung, ASSUME CS, erwähnt, die unter bestimmten Umständen nötig ist. Hier sehen Sie nun ein Beispiel dafür.
Indem wir ASSUME CS benutzen, geben wir dem Assembler eine Falschinformation über den Inhalt des CS-Registers. Diese kleine List ist vollkommen harmlos, solange wir nur kurze Sprünge in unserem Programm

verwenden (in unserem Beispiel JZ). Kurze Sprünge beziehen sich relativ auf den momentanen Programmzählerstand, der aus einer Kombination von CS:IP besteht. Um einen kurzen Sprung durchzuführen, erzeugt der Assembler eine relative Adresse. Diese Adresse ist zu IP relativ, nicht zu CS. Aus diesem Grunde braucht der Assembler eigentlich den Wert von CS gar nicht zu kennen. Wir müssen nur eine Information über CS übergeben, weil der Assembler zu "dickköpfig" ist, um zu erkennen, was er tatsächlich benötigt und was nicht. Hier nun das Beispiel:

```
; TPLOESCH, eine Routine um den Tastaturpuffer zu löschen

SCHNITTSTELLE SEGMENT 'Code'
              PUBLIC  TPLOESCH
              ASSUME  CS:SCHNITTSTELLE
TPLOESCH      PROC    FAR
              PUSH    BP
              MOV     BP,SP
              MOV     AH,1          ;erster Test auf Daten
              INT     22
WHILE:                              ;Schleifenbeginn
              JZ      SHORT RETURN  ;solange ZF = 0
              MOV     AH,0          ;Daten ...
              INT     22            ;... entfernen
              MOV     AH,1          ;wiederholter Test ...
              INT     22            ;... auf Daten
              JMP     SHORT WHILE   ;Schleifenende
RETURN                              ;Rücksprung zum aufrufenden
                                    ;Programm
              POP     BP
              RET
TPLOESCH      ENDP
SCHNITTSTELLE ENDS
              END
```

Kapitel 12
Verschiedene BIOS-Routinen

12.1 Routinen für RS-232 serielle Kommunikation 194
12.1.1 Routine 0: Parameter für seriellen Port initialisieren 195
12.1.2 Routine 1: Ein Zeichen senden 196
12.1.3 Routine 2: Ein Zeichen empfangen 196
12.1.4 Routine 3: Status des seriellen Ports feststellen 197
12.2 Kassettenrecorderroutinen 197
12.2.1 Routine 0: Motor einschalten 198
12.2.2 Routine 1: Motor ausschalten 198
12.2.3 Routine 2: Datenblöcke lesen 198
12.2.4 Routine 3: Datenblöcke schreiben 199
12.3 Erweiterte Routinen beim AT 199
12.4 Druckerroutinen 200
12.4.1 Routine 0: Ein Byte an den Drucker senden 200
12.4.2 Routine 1: Drucker initialisieren 201
12.4.3 Routine 2: Druckerstatus feststellen 201
12.5 Sonstige Routinen 201
12.5.1 Interrupt 5: Bildschirmdruckroutine 202
12.5.2 Interrupt 17: Ausstattung feststellen 202
12.5.3 Interrupt 18: Speicherkapazität feststellen 203
12.5.4 Interrupt 24: ROM-BASIC aktivieren 204
12.5.5 Interrupt 25: Urladerstartroutine aktivieren 204
12.5.6 Interrupt 26: Tageszeitroutine 205
12.5.6.1 Routine 0: Zählerstand lesen 206
12.5.6.2 Routine 1: Zählerstand setzen 206
12.6 Tageszeitroutinen des AT 206

In diesem Kapitel werden ROM-BIOS-Routinen behandelt, die entweder nicht komplex oder nicht wichtig genug sind, um ein eigenes Kapitel zu füllen. Das sind die Routinen für den seriellen Kommunikationsport, die Kassettenrecorderroutinen, die AT-Erweiterungen und die Druckerroutinen. Einige weitere, praktisch bedeutungslose Routinen sind am Ende des Kapitels kurz aufgeführt.

12.1 Routinen für RS-232 serielle Kommunikation

Bevor wir die Routinen für die asynchrone serielle Kommunikation behandeln, müssen einige Begriffe erläutert werden. Es wird vorausgesetzt, daß Sie die Grundlagen der Datenkommunikation beherrschen. Falls Sie im Verlauf des Kapitels Verständnisschwierigkeiten haben, sollten Sie sich eines der zahlreichen Fachbücher über Datenkommunikation zur Hand nehmen.
Für den RS 232-Kommunikationsweg gibt es eine ganze Reihe von Begriffen, einer der gebräuchlichsten ist *Port*. In diesem Zusammenhang hat das Wort *Port* eine andere Bedeutung als bislang. In diesem Buch wird von Ports meist als adressierbaren Wegen gesprochen, die vom Mikroprozessor verwendet werden, um innerhalb des Computers Daten zu transferieren. Die BASIC-Befehle INP und OUT beziehen sich ebenso auf diese Art von Ports wie die Assemblerbefehle IN und OUT. Der RS-232-Port für asynchrone serielle Kommunikation unterscheidet sich davon, er ist ein Vielzweck-E-/A-Port, der zum Anschluß verschiedener Peripheriegeräte an den Computer verwendet werden kann. Typische Beispiele: Ein Modem oder ein Drucker wird über den seriellen Port angeschlossen.
Die Kommunikationsroutinen werden durch Interrupt 20 (hex 14) aufgerufen. Es gibt vier Routinen, die in allen IBM-Modellen vorhanden sind. Sie sind von 0 bis 3 durchnumeriert und werden in Register AH angewählt, indem die entsprechende Routinennummer in das Register geschrieben wird.
Es können bis zu sieben serielle Ports an den PC angeschlossen werden, meist sind es aber nur ein oder zwei. Die Anzahl der verfügbaren Ports wird in Register DX festgehalten. Ist nur ein Port vorhanden, muß in DX eine 0 stehen, für zwei Ports eine 1 usw.

Routine	Beschreibung
0	Parameter für seriellen Port initialisieren
1	Ein Zeichen senden
2	Ein Zeichen empfangen
3	Status des seriellen Ports melden

Tabelle 12-1 Die vier Routinen für den seriellen RS-232 Kommunikationsport, die über Interrupt 29 (hex 14) aufgerufen werden

12.1.1 Routine 0: Parameter für seriellen Port initialisieren

Routine 0 setzt die verschiedenen RS-232-Parameter und initialisiert den seriellen Port. Es werden vier Parameter festgelegt: die *Übertragungsgeschwindigkeit in Baud*, die *Parität*, die *Anzahl der Stop-Bits* und die *Zeichenlänge* (manchmal auch *Wortlänge* genannt). Die Parameter werden in einem 8-bit-Code in Register AL abgelegt. Die Kodierung läßt sich aus untenstehender Tabelle ablesen. Ist die Routine beendet, wird der Kommunikationsstatus genau wie bei Routine 3 (siehe dort) in AX geschrieben.

Bit 7	6	5	4	3	2	1	0	Verwendung
x	x	x	·	·	·	·	·	Baud-Code
·	·	·	x	x	·	·	·	Paritätscode
·	·	·	·	·	x	·	·	Stop-Bit-Code
·	·	·	·	·	·	x	x	Zeichenlängencode

Tabelle 12-2 Die Bit-Kodierung der seriellen Portparameter im Register AL für Routine 0

<u>Anmerkung:</u> **300 Baud war lange Zeit die normale Übertragungsgeschwindigkeit für PCs mit angeschlossenem Modem. Heutzutage werden im allgemeinen 1.200 Baud verwendet, vor allem bei professionellen Anwendungen. Eine Erhöhung auf einen Standard von 2.400 Baud wäre wünschenswert.**

ÜBERTRAGUNGSGESCHWINDIGKEIT IN BAUD

Bit 7	6	5	Wert	Bits pro Sekunde
0	0	0	0	110
0	0	1	1	150
0	1	0	2	300
0	1	1	3	600
1	0	0	4	1,200
1	0	1	5	2,400
1	1	0	6	4,800
1	1	1	7	9,600

PARITÄT

Bit 4	3	Wert	Bedeutung
0	0	0	keine
0	1	1	ungerade Parität
1	0	2	keine
1	1	3	gerade Parität

STOP-BITS

Bit 2	Wert	Bedeutung
0	0	Eins
1	1	Zwei

ZEICHENLÄNGE

Bit 1	Bit 0	Wert	Bedeutung
0	0	0	unbenutzt
0	1	1	unbenutzt
1	0	2	7-bit-Format*
1	1	3	8-bit-Format

* Es gibt nur 128 Standard-ASCII-Zeichen. Diese können im 7-bit Format übertragen werden, so daß das übliche 8-bit-Format nicht benötigt wird.

Tabelle 12-3 Die Bit-Kodierung der vier seriellen Portparameter

12.1.2 Routine 1: Ein Zeichen senden

Routine 1 gibt ein Zeichen auf den seriellen Port aus. Das Zeichen wird in Register AL gegeben, in AH steht das Ergebnis der Operation. Ist AH gleich 0, war der Prozeß erfolgreich, andernfalls wird Bit 7 dieses Registers auf 1 gesetzt. Die anderen Bits in AH melden die Art des Fehlers. Die Kodierung finden Sie bei den Erklärungen zu Routine 3 (Statusroutine).
Es gibt eine Abweichung bezüglich der Fehlermeldung von Routine 1 (gilt auch für Routine 2) gegenüber Routine 3. In den Routinen 1 und 2 wird das erste Bit (Bit 7) verwendet, um generell einen Fehler anzuzeigen. Das Bit kann also nicht mehr zur Anzeige eines Time-out dienen, wie bei Routine 3 erklärt. Daher ist es bei Auftreten eines Fehlers am besten, Routine 3 zu verwenden, um den Status vollständig abzufragen.

12.1.3 Routine 2: Ein Zeichen empfangen

Die Routine 2 ist für das Empfangen von Zeichen über einen bestimmten Kommunikationspfad verantwortlich. Der betreffende Pfad wird in Register DX spezifiziert, das ankommende Zeichen in Register AL abgelegt. Die Routine wartet, bis ein Zeichen oder Signal ankommt, das eine Beendigung veranlaßt. Das kann auch ein Time-out (siehe Routine 3) sein.

AH meldet den Erfolg des Prozesses in Bit 7. Lesen Sie hierzu bitte die Erklärung im letzten Abschnitt (Routine 1) und die Kodierung der Fehlermeldung in Routine 3.

12.1.4 Routine 3: Status des seriellen Ports feststellen

Die Routine 3 meldet den Status des seriellen Kommunikationsports in Register AX. Jedes der 16 Bits steht für ein mögliches Problem. Die Status-Bits werden in zwei Gruppen unterteilt: AH meldet den Leitungsstatus (der auch für die Routinen 1 und 2 von Bedeutung ist) und AL den Modemstatus, soweit zutreffend. Einige Bits signalisieren Fehler, andere nur bestimmte Zustände.

Bits 7	6	5	4	3	2	1	0	Bedeutung (wenn Bit auf 1)	Bits 7	6	5	4	3	2	1	0	Bedeutung (wenn Bit auf 1)
AH-Register (Leitungsstatus)									AL-Register (Modemstatus)								
1	.	.	.	.	.	.	.	Time-out-Fehler	1	.	.	.	.	.	.	.	Empfangsleitungssignal (RLSD)
.	1	.	.	.	.	.	.	Transfer-Shift-Register leer	.	1	.	.	.	.	.	.	Ringkomplement (RI)
.	.	1	.	.	.	.	.	Transfer Halte-Register leer	.	.	1	.	.	.	.	.	Datenendgerät bereit (DSR)
.	.	.	1	.	.	.	.	Unterbrechungsfehler	.	.	.	1	.	.	.	.	Sendebereit (CTS)
.	.	.	.	1	.	.	.	Framing-Fehler	.	.	.	.	1	.	.	.	Delta Empfangsleitungssignal (DRLSD)
.	.	.	.	.	1	.	.	Paritätsfehler	.	.	.	.	.	1	.	.	Abschließende Flanke (TERI)
.	.	.	.	.	.	1	.	Überlauffehler	.	.	.	.	.	.	1	.	Delta Datenendgerät bereit (DDSR)
.	.	.	.	.	.	.	1	Daten bereit	.	.	.	.	.	.	.	1	Delta Sendebereit (DCTS)

Tabelle 12-4 Die Bit-Kodierung der Status-Bytes im Register AX, die von Routine 3 verwendet wird

Anmerkung zum Time-out: Ein Time-out-Fehler liegt vor, wenn die maximale Antwortzeit eines Gerätes überschritten wurde. Die erste Version des ROM-BIOS hatte einen Programmierfehler, der in allen weiteren Versionen nicht mehr auftaucht: Ein serieller Time-out wurde mit der Bitfolge 01010000 (Transfer-Shift-Register leer und Unterbrechungsfehler) statt 10000000 gemeldet. Viele Kommunikationsprogramme berücksichtigen diesen BIOS-Fehler nicht.

12.2 Kassettenrecorderroutinen

Die Kassettenrecorderroutinen werden benötigt, um einen Kassettenrecorder als Datenspeichergerät zu verwenden. Der Recorderanschluß ist nur bei wenigen PC-Modellen wie z.B. dem Original-PC vorhanden. Der XT, der tragbare PC, der AT, der 3270-PC und andere verfügen nicht über einen Recorderanschluß. Ursprünglich war IBM von der Annahme ausgegangen, daß viele PC-Benutzer ihre Daten und Programme auf normalen Musikkassetten speichern wollen, da diese Methode preiswerter als z.B. die Speicherung auf Disketten ist. Einen Kassettenrecorder hat im Unter-

schied zu einem Diskettenlaufwerk fast jeder zuhause. Da die Datenspeicherung auf Kassetten aber sehr langsam und umständlich ist, wird sie ausschließlich im Zusammenhang mit Homecomputern der untersten Preisklasse eingesetzt. Die Annahme der IBM, daß auch PC-Benutzer von dieser Möglichkeit Gebrauch machen könnten, erwies sich als falsch. Dennoch wird der Kassettenrecorderport von ROM-BIOS und BASIC unterstützt. Es ist nicht zu empfehlen, den Recorderanschluß zu nutzen, es sei denn für spezielle Anwendungen wie z.B. als seriellen Kommunikationsport für den Eigengebrauch.
Die Kassettenrecorderroutinen werden über Interrupt 21 (hex 15) aufgerufen. Die Unterroutine (0 bis 3) wird wie immer in Register AH festgelegt.

Routine	Beschreibung
0	Motor einschalten
1	Motor ausschalten
2	Datenblöcke lesen
3	Datenblöcke schreiben

Tabelle 12-5 Die vier Kassettenrecorderroutinen im ROM-BIOS, die über Interrupt 21 aufgerufen werden können

12.2.1 Routine 0: Motor einschalten

Die Routine 0 schaltet den Kassettenrecordermotor ein. Dies geschieht bei einem Recorder im Unterschied zu einem Diskettenlaufwerk nicht vollautomatisch. Ein Programm, das diese Routine verwendet, sollte mit einer Verzögerung arbeiten, um den Start des Motors abzuwarten, bevor Zugriffe erfolgen.

12.2.2 Routine 1: Motor ausschalten

Die Routine 1 schaltet den Motor wieder aus, auch hier existiert keine automatische Funktion wie für das Ausschalten des Diskettenlaufwerkmotors.

12.2.3 Routine 2: Datenblöcke lesen

Routine 2 liest einen oder mehrere Datenblöcke von Kassette. Ein Datenblock auf einer Kassette hat eine Standardlänge von 256 Bytes, ähnlich wie Disketten 512 Bytes pro Sektor verwenden. Die Anzahl der zu lesenden Bytes wird im CX-Register spezifiziert. Obwohl die Daten in 256-

byte-Blöcken untergebracht sind, kann jede beliebige Datenmenge auf Band geschrieben oder von Band gelesen werden. Der Wert im CX-Register muß kein Vielfaches von 256 sein. Das Registerpaar ES:BX wird als Zeiger verwendet. Es deutet auf die Stelle im Speicher, an der die Daten abgelegt werden sollen.
Ist die Routine beendet, enthält das DX-Register die Anzahl der gelesenen Bytes. ES:BX zeigt auf das Byte, das unmittelbar hinter dem zuletzt transferierten Byte liegt. Die Carry-Flagge (CF) ist auf 0 oder 1 gesetzt und zeigt somit Erfolg oder Mißerfolg der Operation an. Im Falle einer Störung ist in Register AH die Fehlerart abgelegt.

Code	Bedeutung
1	Prüfsummenfehler ("Cyclical Redundancy Check" oder CRC)
2	Datenübergänge fehlerhaft: Bit-Signale beschädigt
3	Daten auf Kassette nicht gefunden

Tabelle 12-6 Die Bit-Kodierung im Register AH, wenn CF einen Fehler meldet

12.2.4 Routine 3: Datenblöcke schreiben

Routine 3 schreibt einen oder mehrere Datenblöcke mit einer Länge von 256 Bytes auf Band. Analog zu Routine 2 steht die Anzahl der zu schreibenden Bytes im CX-Register und das Paar ES:BX zeigt auf den Datenbereich im Speicher. Falls die Anzahl der Datenbytes kein Vielfaches von 256 ist, wird der letzte Block bis auf die volle Länge aufgefüllt.
Ist die Routine beendet, sollte das Register CX auf 0 und das Registerpaar ES:BX direkt auf die Speicherstelle hinter dem zuletzt geschriebenen Byte gesetzt werden.
Bei dieser Routine ist kein Fehlerstatus abfragbar, weil normale Kassettenrecorder keine Fehlermeldungen übermitteln können. Alle geschriebenen Daten sollten daher noch einmal gelesen werden, um eventuelle Fehler zu entdecken.

12.3 Erweiterte Routinen beim AT

Mit dem AT wurden auch einige neue BIOS-Routinen eingeführt. Sie werden wie die Kassettenrecorderroutinen über den Interrupt 21 (hex 15) aufgerufen. Die Unterroutinen besitzen Nummern von hex 80 bis hex 91 und werden im Register AH spezifiziert. Wir werden die Routinen an dieser Stelle nicht im Detail behandeln. Falls Sie daran interessiert sind, lesen Sie bitte in Kapitel 13 nach.

Routine (hex)	Beschreibung
80	Schnittstelle offen
81	Schnittstelle geschlossen
82	Programmende
83	Auf Ereignis warten
84	Joystick
85	SysReg-Tastendruck
86	Warten
87	Block verlagern
88	Erweiterte Speicherkapazität
89	Auf virtuellen Speicher schalten (VORSICHT: vor Verwendung bitte das BIOS-Listing aufmerksam lesen)
90	Gerät-beschäftigt-Schleife
91	Flagge setzen und Interrupt beenden

Tabelle 12-7 Die 12 neuen Routinen, die im AT zur Verfügung stehen (der Aufruf erfolgt über Interrupt 21)

12.4 Druckerroutinen

Die Druckerroutinen des ROM-BIOS unterstützen die Ausgabe von Daten auf den Drucker. Beim Standard-PC beziehen sich die Routinen auf den Paralleldruckeranschluß, bei einigen PC-Modellen kann die Ausgabe auf einen seriellen Port umgelenkt werden.
Die Druckerroutinen werden über den Interrupt 23 (hex 17) aufgerufen, die Auswahl der drei Unterroutinen (Numerierung von 0 bis 2) geschieht in Register AH. Die Architektur des PC erlaubt den Anschluß mehrerer Drucker, wobei die Nummer des jeweils aktuellen Druckers für alle Routinen in Register DX spezifiziert werden sollte. Für die Bildschirmdruckroutine wird automatisch der Drucker 0 verwendet. Näheres hierzu finden Sie in Kapitel 12.5.1.

Routine	Beschreibung
0	Ein Byte an den Drucker senden
1	Drucker initialisieren
2	Druckerstatus feststellen

Tabelle 12-8 Die drei Druckerroutinen des ROM-BIOS, die mit Interrupt 21 (hex 17) aufgerufen werden können

12.4.1 Routine 0: Ein Byte an den Drucker senden

Routine 0 sendet ein Byte, das in Register AL steht, an den Drucker. Ist die Routine beendet, wird in AH der Druckerstatus gesetzt (siehe Routine 2). Hinweise zur Unterbrechung des Druckvorgangs finden Sie bei der Erläuterung der Routine 2.

12.4.2 Routine 1: Drucker initialisieren

Routine 1 initialisiert den Drucker, indem zwei Kontrollcodes an den Druckerkontrollport (üblicherweise hex 08 und hex 0C) gesendet werden. Wie bei den anderen beiden Druckerroutinen wird der Status in Register AH abgelegt.

12.4.3 Routine 2: Druckerstatus feststellen

Routine 2 meldet den Druckerstatus in das AH-Register.

			Bit					
7	6	5	4	3	2	1	0	Bedeutung (wenn auf 1 gesetzt)
1	.	.	.	.	.	.	.	Kein Druckvorgang im Ablauf (nicht "busy")
.	1	.	.	.	.	.	.	Bestätigung vom Drucker
.	.	1	.	.	.	.	.	Papierzufuhr gestört
.	.	.	1	.	.	.	.	Drucker ausgewählt
.	.	.	.	1	.	.	.	E/A-Fehler
.	.	.	.	.	1	.	.	unbenutzt
.	.	.	.	.	.	1	.	unbenutzt
.	.	.	.	.	.	.	1	Time-out

Tabelle 12-9 Die Druckerstatus-Bits im Register AH, aufgerufen von Routine 2

Der Drucker-Time-out-Fehler bereitet oftmals Probleme. Jeder E/A-Treiber muß ein Zeitlimit für die Anwort des kontrollierten Gerätes setzen. Die Zeitspanne sollte nicht übermäßig lang sein, damit ein nicht antwortendes Gerät sofort gemeldet werden kann. Unglücklicherweise gibt es aber eine Druckerfunktion, die eine längere Zeitspanne umfaßt: der Seitenvorschub. Die dafür bereitgestellte Zeit variiert bei den verschiedenen BIOS-Versionen.

12.5 Sonstige Routinen

Das ROM-BIOS enthält einige Routinen, die sich in keine Kategorie einordnen lassen.

Interrupt		
Dez	Hex	Beschreibung
5	5	Bildschirmdruckroutine (Bildschirminhalt ausdrucken)
17	11	Ausstattung feststellen
18	12	Speicherkapazität feststellen
24	18	ROM-BASIC aktivieren
25	19	Urladerstartroutine aktivieren
26	1A	Tageszeitroutine

Tabelle 12-10 Sechs ROM-BIOS-Routinen für unterschiedliche Zwecke und die jeweiligen Interrupts

12.5.1 Interrupt 5: Bildschirmdruckroutine

Der Interrupt 5, der die Bildschirmdruckroutine aufruft, wird ausgeführt, wenn die Taste PrtSc (*Print Screen*) zusammen mit einer Umschalttaste betätigt wird. Die Routine kann auch von Programmen benutzt werden.
Die Routine läßt die Cursorposition auf dem Bildschirm unverändert und druckt alle sichtbaren Zeichen des Bildschirmes sowohl in den Text- als auch in den Grafik-Modi aus. Sie greift dabei auf die Standardbildschirmroutinen zurück (zum Bewegen des Cursor über den Bildschirm und zum Lesen der Daten aus dem Bildschirmpuffer) und verwendet die Standarddruckerroutinen.
Alle Ausgaben der Routine erfolgen über den Standarddrucker 0. Ein- und Ausgaberegister gibt es bei der Routine nicht, aber an der Speicherstelle hex 500 im unteren Speicherbereich (siehe Kapitel 3.2.2) wird ein Statuscode abgelegt. Steht dort ein Wert von 255 (hex FF), bedeutet das, daß eine frühere Bildschirmdruckoperation nicht erfolgreich ausgeführt werden konnte. Der Wert 0 zeigt an, daß kein Fehler aufgetreten ist und der Wert 1, daß gerade eine Bildschirmdruckoperation ausgeführt wird. Während des Druckvorganges kann keine zweite Druckoperation eingeleitet werden. Eine Druckerwarteschlange existiert für diese Routine nicht.

12.5.2 Interrupt 17 (hex 11): Ausstattung feststellen

Die Routine 17 lädt eine Liste der angeschlossenen Hardwareausstattung in das Register AX. Die gleichen Informationen finden Sie auch im unteren Speicherbereich bei Speicherstelle hex 410 (lesen Sie bitte in Kapitel 3.2.2 nach). Die Liste befindet sich in den Bits eines 2-byte-Wortes. Die Routine ist eine Ergänzung zu Interrupt 18.
Die Ausstattungsliste wird nur beim Startprozeß einmal in den Speicher geladen und bleibt dann unverändert. Die Liste läßt sich aber mit Software manipulieren. Wir können z.B. einige Peripheriegeräte abhängen, so

daß sie nicht verwendet werden können, indem sie aus der Liste herausgenommen werden. Veränderungen der Ausstattungsliste wirken nicht in jedem Fall so, wie man es erwarten würde: Manche Manipulationen bleiben wirkungslos. Unter Interrupt 25 (wird gegen Ende des Kapitels erläutert) finden Sie Informationen, wie Sie mit der Ausstattungsliste umzugehen haben, um verläßliche Ergebnisse zu erhalten.
Das Format der Ausstattungsliste wurde mit dem Erscheinen des Original-PC festgelegt. Das hat zur Folge, daß einige Teile der Liste für andere Modelle keine oder eine abweichende Bedeutung haben.

							Bits								
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Bedeutung
x	x	.	.	.	.	.	.	.	.	.	.	.	.	.	.	Anzahl der Drucker
.	.	x	.	.	.	.	.	.	.	.	.	.	.	.	.	Serieller Drucker: 1 = vorhanden (nur PCjr)
.	.	.	x	.	.	.	.	.	.	.	.	.	.	.	.	Spieleadapter: 1 = vorhanden
.	.	.	.	x	x	x	.	.	.	.	.	.	.	.	.	Anzahl der RS-232 Ports
.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	DMA-Chip: 0 = vorhanden
.	.	.	.	.	.	.	.	x	x	.	.	.	.	.	.	+ 1 = Anzahl der Diskettenlaufwerke, 0 = ein Laufwerk (siehe Bit 0)
.	.	.	.	.	.	.	.	.	.	x	x	.	.	.	.	Bildschirmmodus: 10 = 80-Spalten-Farbe, 11 = Monochrom (XT/PC)
.	.	.	.	.	.	.	.	.	.	.	.	x	x	.	.	Speicherkapazität auf der Hauptplatine 11 = 64 Kbyte
.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	unbenutzt (auf 0 gesetzt)
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	1, falls Diskettenlaufwerk vorhanden (siehe Bits 6 und 7)

Tabelle 12-11 Die Bit-Kodierung der Ausstattungsliste, die von Interrupt 17 (hex 11) in Register AX geladen wird

12.5.3 Routine 18 (hex 12): Speicherkapazität feststellen

Interrupt 18 ruft eine Routine auf, die die verfügbare Speicherkapazität in Kbyte in das AX-Register schreibt. Dieselbe Information finden wir in Speicherstelle hex 413 im unteren Speicherbereich (siehe Kapitel 3.2.2). Diese Routine ist eine Ergänzung zu Interrupt 17.

In den Standard-PC-Modellen ist die Speicherkapazität den entsprechenden Schalterstellungen auf der Platine zu entnehmen. Unter gewissen Umständen wird durch die Routine weniger Speicherkapazität angezeigt, als tatsächlich vorhanden ist.

12.5.4 Interrupt 24 (hex 18): ROM-BASIC aktivieren

Mit Interrupt 24 wird im allgemeinen das ROM-BASIC aktiviert. In der Praxis ist das bedeutungslos, da das ROM-BASIC kaum benutzt wird.

12.5.5 Interrupt 25 (hex 19): Urladerstartroutine aktivieren

Interrupt 25 aktiviert die Startroutine des Computers und führt fast zum gleichen Resultat wie die Tastenkombination Ctrl-Alt-Del. Der Urladerstartinterrupt übergeht aber den Speichertest der Einschaltroutine und die Reset-Operation von Ctrl-Alt-Del.
Es gibt zwei Anwendungen für den Interrupt 25, die bekannt sind. Die eine ist die Herbeiführung eines "Abstürzens" des Computers, wenn eine nicht tolerierbare Situation eintritt, z.B. die offensichtliche Verletzung des Kopierschutzes. Viele kopiergeschützte Programme "wehren" sich auf diese Weise gegen einen Kopierversuch.
Die andere Anwendung liegt im Neustarten des Computers, ohne die Reset-Operation und die Speicherabfrage nochmals zu durchlaufen. Der Interrupt ist also für Programme, die die Interrupts 17 und 18 verwenden, sehr nützlich: Wollen wir die Ausstattungsliste oder die Speichergröße (z.B. zum Reservieren eines Speicherbereiches für eine RAM-Disk) manipulieren, können wir uns nicht bei allen Programmen darauf verlassen (vor allem nicht bei DOS), daß die Speicher- und die Ausstattungsspezifikationen ständig neu kontrolliert werden. Ein Programm kann einen Teil des Speicherbereiches reservieren, die Kapazitätsangabe verändern und dann diesen Interrupt aufrufen, um das System erneut zu starten. Wird diese Prozedur durchlaufen und DOS aktiviert, nimmt DOS die neu festgelegte Speicherkapazität als gegeben an. Weder DOS noch die normalen DOS-Programme erkennen, daß ihnen Speicherplatz vorenthalten wurde, Störungen treten nicht auf.
Hier ein kurzes Beispiel, ein Auszug aus einen Assemblerprogramm, bei dem der BIOS-Eintrag der Speichergröße verändert und anschließend Interrupt 25 aufgerufen wird, um das System neu zu starten:

```
MOV   AX,40H              ;BIOS-Datensegment hex 40 ...
MOV   ES,AX               ;... nach ES bringen
MOV   WORD PTR ES:19,256  ;Speicher auf 256 Kbyte setzen
INT   25                  ;System neu starten
```

12.5.6 Interrupt 26 (hex 1A): Tageszeitroutine

Über den Interrupt 26 können die Uhrfunktionen angesprochen werden. Es stehen zwei Routinen (0 und 1) zur Verfügung. Die Routinennummer wird in Register AH festgelegt.

Routine	Beschreibung	Register
0	Zählerstand lesen	CX = höherwertiger Teil des Zählerstandes DX = niederwertiger Teil des Zählerstandes AL = 0, wenn noch keine ganze 24-Stunden-Periode vorüber ist AL <> 0, falls neuer Tag
1	Zählerstand setzen	CX = höherwertiger Teil des Zählerstandes DX = niederwertiger Teil des Zählerstandes

Tabelle 12-12 Die zwei Tageszeitroutinen des Interrupt 26

Das ROM-BIOS berechnet die aktuelle Tageszeit, indem es von einem Anfangswert ausgehend die Impulse zählt, die durch den Systemtakt erzeugt werden. Die Zählung beginnt bei Mitternacht. Die Impulse bestehen aus dem Interrupt 8, der in periodischen Abständen von einem Oszillator aufgerufen wird. Jeder von Interrupt 8 ankommende Impuls bewirkt eine Erhöhung des Zählerstandes um eins. Wenn der Wert für 24 Stunden erreicht ist, wird ein Eintrag vorgenommen, daß das "Ereignis Mitternacht" stattgefunden hat und der Zählerstand auf 0 zurückgesetzt. Es gibt keine Möglichkeit, festzustellen, ob das "Ereignis Mitternacht" mehrere Male aufgetreten ist.
Die Hauptoszillator schwingt mit einer Frequenz von genau (1.193.180 / 64 Kbyte) Hertz oder ungefähr 18,2 mal in der Sekunde. Der Zählerstand wird in der unteren Speicherstelle hex 46C gespeichert, er besteht aus einem 4-byte-Ganzzahlwort. Der Wert, der bei Mitternacht erreicht ist, beträgt 1.573.040 (hex 1800B0). Er wird zum Vergleich mit dem laufenden Zählerstand benötigt und ist in hex 470 (siehe Kapitel 3.2.2) abgelegt. Wenn DOS die aktuelle Zeit benötigt, liest es den Zählerstand und errechnet die Zeit aus dem Wert. Gleichzeitig wird eine Abfrage nach dem "Ereignis Mitternacht" durchgeführt und je nach Ergebnis das Datum erhöht. Die Tageszeit kann mit folgenden BASIC-Formeln aus dem Zählerstand abgeleitet werden:

```
STUNDE = ZAEHLER \ 65543 (hex 10007)
REST = ZAEHLER MOD 65543
MINUTEN = REST \ 1092 (hex 444)
REST = REST MOD 1092
SEKUNDEN = REST \ 18.21        'genauer Wert; sonst 18
REST = REST MOD 18.21
HUNDERTSTEL = CINT( REST * 100)
```

Mit folgender Formel läßt sich der Zählerstand aus der Zeit berechnen:

```
ZAEHLER = (STUNDE * 65543.33) + (MINUTEN * 1092.38)
+ (SEKUNDEN * 18.21) + (HUNDERTSTEL * .182)
```

12.5.6.1 Routine 0: Zählerstand lesen

Die Routine 0 übergibt den Zählerstand an zwei Register: der höherwertige Teil wird in CX, der niederwertige in DX abgelegt. Ist das Register AL gleich 0, wurde seit dem letzten Lesen oder Setzen des Zählerstandes das "Ereignis Mitternacht" nicht erreicht. Andernfalls ist AL gleich 1. Das Mitternachtssignal wird immer zurückgesetzt (auf 0), wenn der Zählerstand gelesen wird. Jedes Programm, das die Routine verwendet, muß das Mitternachtssignal lesen und gegebenenfalls das Datum aktualisieren. In DOS-Programmen sollte die Routine nicht direkt verwendet werden, um den mit der Datumsberechnung verbundenen Aufwand zu vermeiden.

12.5.6.2 Routine 1: Zählerstand setzen

Routine 1 setzt den Zählerstand in der Speicherstelle hex 46C. Der Wert wird dem Registerpaar CX:DX entnommen. Das Mitternachtssignal wird jedesmal zurückgesetzt, wenn der Wert die 24-Stunden-Grenze erreicht.

12.6 Tageszeitroutinen des AT

Die Unterroutinen 2 bis 6 wurden mit der BIOS-Version des AT neu eingeführt. Sie werden mit dem Interrupt 26 aufgerufen. Die Unterroutinen 2, 3 und 4 lesen und stellen die Echtzeituhr, die Unterroutinen 5 und 6 setzen einen Alarmzeitpunkt (bis zu 24 Stunden ab der aktuellen Zeit). Nähere Information über die Routinen finden Sie in Kapitel 13 und in der BIOS-Auflistung des Technischen Referenzhandbuches zum AT.

Kapitel 13
Zusammenfassung: BIOS-Routinen

13.1 Kurzzusammenfassung 208
13.2 Ausführliche Zusammenfassung 210

13.1 Kurzzusammenfassung

Art	Interrupt Dez	Interrupt Hex	Routinennummer (hex)	Erklärung	Modell
Bildschirmdruck	5	5	–	Bildschirminhalt ausdrucken	
Bildschirm	16	10	0	Bildschirmmodus festlegen	
Bildschirm	16	10	1	Cursorgröße festlegen	
Bildschirm	16	10	2	Cursorposition setzen	
Bildschirm	16	10	3	Cursorposition abfragen	
Bildschirm	16	10	4	Lichtgriffelposition abfragen	
Bildschirm	16	10	5	Aktive Azeigeseite festlegen	
Bildschirm	16	10	5 (AL:128)	Anzeigenseitenregister melden	
Bildschirm	16	10	5 (AL:129)	CPU-Anzeigeseitenregister setzen	
Bildschirm	16	10	5 (AL:130)	Bildschirmanzeigeseitenregister setzen	
Bildschirm	16	10	5 (AL:131)	Beide Anzeigenregister setzen	
Bildschirm	16	10	6	Fenster nach oben rollen	
Bildschirm	16	10	7	Fenster nach unten rollen	
Bildschirm	16	10	8	Zeichen und Attribut an der Cursorposition lesen	
Bildschirm	16	10	9	Zeichen und Attribut an der Cursorposition schreiben	
Bildschirm	16	10	A	Zeichen an der Cursorposition schreiben	
Bildschirm	16	10	B	Farbpalette festlegen	
Bildschirm	16	10	C	Pixel setzen	
Bildschirm	16	10	E	TTY-Zeichen schreiben	
Bildschirm	16	10	F	Bildschirmmodus feststellen	
Bildschirm	16	10	13	Zeichenkette (String) schreiben	AT
Ausstattung	17	11	–	Ausstattung feststellen	
Speicher	18	12	–	Speicherkapazität feststellen	
Laufwerk	19	13	0	Rücksetzen der Diskettenstation	
Laufwerk	19	13	1	Laufwerkstatus feststellen	
Laufwerk	19	13	2	Sektoren lesen	
Laufwerk	19	13	3	Sektoren schreiben	
Laufwerk	19	13	4	Sektoren verifizieren	
Laufwerk	19	13	5	Spur formatieren	
Laufwerk	19	13	8	Aktuelle Laufwerksparameter lesen	AT
Laufwerk	19	13	9	Festplattenparametertabelle initialisieren	AT
Laufwerk	19	13	A	Lange Sektoren lesen	AT
Laufwerk	19	13	B	Lange Sektoren schreiben	AT
Laufwerk	19	13	C	Zylinder anfahren	AT
Laufwerk	19	13	D	Alternativer Laufwerk-Reset	AT
Laufwerk	19	13	10	Test, ob Laufwerk bereit	AT
Laufwerk	19	13	11	Laufwerk neu kalibrieren	AT
Laufwerk	19	13	14	Controller-Diagnose	AT
Laufwerk	19	13	15	Laufwerkstyp feststellen	AT
Laufwerk	19	13	16	Diskettenwechsel erkennen	AT
Laufwerk	19	13	17	Diskettenlaufwerk festlegen	AT
Serieller Port	20	14	0	Parameter für seriellen Port initialisieren	
Serieller Port	20	14	1	Ein Zeichen senden	
Serieller Port	20	14	2	Ein Zeichen empfangen	
Serieller Port	20	14	3	Status des seriellen Ports feststellen	
Recorder	21	15	0	Motor einschalten	
Recorder	21	15	1	Motor ausschalten	
Recorder	21	15	2	Datenblöcke lesen	
Recorder	21	15	3	Datenblöcke schreiben	
Schnittstelle	21	15	80	Schnittstelle offen	AT

Art	Interrupt Dez	Hex	Routinennummer (hex)	Erklärung	Modell
Schnittstelle	21	15	81	Schnittstelle geschlossen	AT
Schnittstelle	21	15	82	Programmende	AT
Schnittstelle	21	15	83	Auf Ereignis warten	AT
Joystick	21	15	84	Joystick	AT
System	21	15	85	SysReg-Tastendruck	AT
Schnittstelle	21	15	86	Warten	AT
Schnittstelle	21	15	87	Block verlagern	AT
Speicher	21	15	88	Erweiterte Speicherkapazität melden	AT
Speicher	21	15	89	Auf virtuellen Speicher schalten	AT
Schnittstelle	21	15	90	Gerät-beschäftigt-Schleife	AT
Schnittstelle	21	15	91	Flagge setzen und Interrupt beenden	AT
Tastatur	22	16	0	Zeichen des nächsten Tastendruckes holen	
Tastatur	22	16	1	Meldung ob Zeichen bereit	
Tastatur	22	16	2	Umschaltstatus feststellen	
Drucker	23	17	0	Ein Byte an den Drucker senden	
Drucker	23	17	1	Drucker initialisieren	
Drucker	23	17	2	Druckerstatus feststellen	
BASIC	24	18	–	ROM-BASIC aktivieren	
Urlader	25	19	–	Urladerstartroutine aktivieren	
Uhr	26	1A	0	Zählerstand lesen	
Uhr	26	1A	1	Zählerstand setzen	
Uhr	26	1A	2	Uhrzeit ablesen	AT
Uhr	26	1A	3	Uhrzeit stellen	AT
Uhr	26	1A	4	Datum ablesen	AT
Uhr	26	1A	5	Datum stellen	AT
Uhr	26	1A	6	Alarm setzen	AT
Uhr	26	1A	7	Alarm rücksetzen	AT

13.2 Ausführliche Zusammenfassung

Routine	Interrupt (hex)	Register Eingabe	Ausgabe	Erläuterung
Bildschirm-drucken	05	AH = 05	–	Bildschirminhalt ausdrucken. Status im unteren Speicherbereich bei hex 500 (0050 : 0000)
Bildschirmroutinen				
Bildschirmmodus festlegen	10	AH = 00 AL = Bildschirm-modus	– –	Bildschirmmodus in AL: 00: 40 × 25 Zeichen, 16 Grautöne 01: 40 × 25 Zeichen, 16/8 Farben 02: 80 × 25 Zeichen, 16 Grautöne 03: 80 × 25 Zeichen, 16/8 Farben 04: 320 × 200 Pixel, 4 Farben 05: 320 × 200 Pixel, 4 Grautöne 06: 640 × 200 Pixel, s/w 07: 80 × 25 Zeichen, s/w 08: 160 × 200 Pixel, 16 Farben 09: 320 × 200 Pixel, 16 Farben 0A: 640 × 200 Pixel, 4 Farben
Cursorgröße festlegen	10	AH = 01 CH = Anfangsrasterzeile CL = Endrasterzeile	–	Farb-/Grafikadapter nutzt Rasterzeilen 0–7 Monochromadapter nutzt Rasterzeilen 0–13
Cursorposition setzen	10	AH = 02 BH = Nummer der aktiven Anzeigeseite DH = Zeile DL = Spalte		
Cursorposition abfragen	10	AH = 03 BH = Nummer der aktiven Anzeigeseite	CH = Anfangsrasterzeile CL = Endrasterzeile DH = Zeile DL = Spalte	
Lichtgriffel-position abfragen	10	AH = 04	AH = Auslösesignal (1 = ausgelöst) BX = Pixelspalte CH = Pixelzeile DH = Zeichenzeile DL = Zeichenspalte	
Aktive Anzeigeseite festlegen	10	AH = 05 AL = Seitennummer		
Anzeigeseiten-register melden	10	AH = 05 AL = 80	BH = Bildschirm-seitenregister BL = CPU-Seitenregister	
CPU-Anzeige-seitenregister setzen	10	AH = 05 AL = 81 BL = CPU-Seiten-register	BH = Bildschirmseiten-register BL = CPU-Seitenregister	

Routine	Interrupt (hex)	Register Eingabe	Ausgabe	Erläuterung
Bildschirmanzeige-seitenregister setzen	10	AH = 05 AL = 82 BH = Bildschirmseiten-register	BH = Bildschirmseitenregister BL = CPU-Seitenregister	
Beide Anzeige-setzen	10	AH = 05 AL = 83 BH = Bildschirmseiten-register BL = CPU-Seiten-register	BH = Bildschirmseitenregister BL = CPU-Seitenregister	
Fenster nach oben rollen	10	AH = 06 AL = Anzahl der zu rollenden Zeilen BH = Anzeigeattribut der Leerzeilen CH = obere Zeile CL = linke Spalte DH = untere Zeile DL = rechte Spalte	–	
Fenster nach unten rollen	10	AH = 07 AL = Anzahl der zu rollenden Zeilen BH = Anzeigeattribut der Leerzeilen CH = obere Zeile CL = linke Spalte DH = untere Zeile DL = rechte Zeile		
Zeichen und Attribut an der Curorposition lesen	10	AH = 08 BH = Anzeigeseite	AH = Zeichen AL = Attribut	
Zeichen und Attribut an der Cursorposition schreiben	10	AH = 09 AL = Zeichen BH = Seitennummer BL = Attribut CX = Zeichenanzahl	–	
Zeichen an der Cursorposition schreiben	10	AH = 0A AL = Zeichen BH = Seitennummer BL = Farbe (in Grafik-modi) CX = Zeichenanzahl		
Farbpalette festlegen	10	AH = 0B BH = Farbpalettenwert BL = Palettenauswahl	–	

Routine	Interrupt (hex)	Register Eingabe	Ausgabe	Erläuterung
Pixel setzen	10	AH = 0C AL = Farbe CX = Pixelspalte DL = Pixelzeile	–	
Pixel abfragen	10	AH = 0D CX = Pixelspalte DL = Pixelzeile	AL = Farbcode	
TTY-Zeichen schreiben	10	AH = 0E AL = Zeichen BL = Farbe für Grafikmodi	–	
Bildschirmmodus feststellen	10	AH = 0F	AH = Anzahl der Zeichen pro Zeile AL = Bildschirmmodus BH = Seitennummer	
AT Zeichenkette (String) ohne Cursorbewegung schreiben	10	AH = 13 AL = 00 BL = Attribut BH = Seitennummer DX = Anfangscursorposition CX = Stringlänge (Anzahl Zeichen) ES:BP = Zeiger auf den Stringanfang		
AT Zeichenkette (String) mit Cursorbewegung schreiben	10	AH = 13 AL = 01 BL = Attribut BH = Seitennummer DX = Anfangscursorposition CX = Stringlänge (Anzahl Zeichen) ES:BP = Zeiger auf den Stringanfang		
AT Zeichenkette mit individuellen Attributen ohne Cursorbewegung schreiben	10	AH = 13 AL = 02 BH = Seitennummer DX = Anfangscursorposition CX = Stringlänge (Anzahl Zeichen) ES:BP = Zeiger auf den Stringanfang	–	

Routine	Interrupt (hex)	Register Eingabe	Register Ausgabe	Erläuterung
Ausstattungsroutinen				
Ausstattung feststellen	11	–	A = bitkodierte Ausstattungsliste	Bits in AX: 00 = Laufwerk 01 = Arithmetik-Coprozessor 02, 03 = RAM auf der Hauptplatine in 16-Kbyte-Blöcken 04,05 = Anfangsbildschirmmodus: 00 = unbenutzt; 01 = 40 × 25 Farbe; 10 = 80 × 25 Farbe; 11 = 80 × 25 s/w. 06, 07 = Anzahl der Laufwerke 08 = DMA möglich? 00 = ja; 01 = nein 09, 10, 11 = Anzahl RS-232-Karten 12 = Spielanschluß vorhanden (beim AT unbenutzt) 13 = serieller Drucker angeschlossen 14, 15 = Anzahl der Drucker
Speicherroutinen				
Speicherkapazität feststellen	12	–	AX = Speicherkapazität (in Kbyte)	
Laufwerkroutinen				
Rücksetzen der Diskettenstation	13	AH = 00	–	
Diskettenststatus feststellen	13	AH = 01	AL = Statuscode	Statuswert: AL = 1: Ungültiger Befehl AL = 2: Sektorkennung ungültig oder nicht auffindbar AL = 3: Schreibschutzfehler AL = 4: Sektor nicht auffindbar AL = 6: Diskette entfernt AL = 8: DMA-Fehler AL = 9: Überschreitung der DMA-Grenze von 64 Kbyte AL = 10: CRC-Fehler AL = 20: Controller-Fehler AL = 40: Anfahren der Spur unmöglich AL = 80: Time-out-Fehler
Sektoren lesen	13	AH = 02 AL = Anzahl Sektoren CH = Sparnummer CL = Sektornummer DH = Kopfnummer DL = Laufwerksnummer ES:BX = Zeiger auf Puffer	CF = Statussignal AH = Statuscode AL = Anzahl der gelesenen Sektoren	Statuscode in AH: s. Laufwerksroutine 1 (Diskettenstatus feststellen)

Routine	Interrupt (hex)	Register Eingabe	Ausgabe	Erläuterung
Sektoren schreiben	13	AH = 3 AL = Anzahl Sektoren CH = Spurnummer CL = Sektornummer DH = Kopfnummer DL = Laufwerksnummer ES:BX = Zeiger auf Puffer	CF = Statussignal AH = Statuscode AL = Anzahl der geschriebenen Sektoren	Statuscode in AH: s. Laufwerksroutine 1 (Diskettenstatus feststellen)
Sektoren verifizieren	13	AH = 04 AL = Anzahl Sektoren CH = Sparnummer CL = Sektornummer DH = Kopfnummer DL = Laufwerksnummer	CF = Statussignal AH = Statuscode AL = Anzahl der geprüften Sektoren	Statuscode in AH: s. Laufwerksroutine 1 (Diskettenstatus feststellen)
Spur formatieren	13	AH = 05 AL = Anzahl Sektoren CH = Spurnummer CL = Sektornummer DH = Kopfnummer DL = Laufwerksnummer ES:BX = Zeiger auf 4-byte-Feld: Byte 1 = Spur Byte 2 = Kopf Byte 3 = Sektor Byte 4 = Bytes/Sektor	CF = Statusflagge AH = Statuscode	Statuscode in AH: s. Laufwerksroutine 1 (Diskettenstatus feststellen)
AT Aktuelle Laufwerksparameter feststellen	13	AH = 08	DL = Anzahl Laufwerke DH = max. Anzahl Seiten CL = max. Anzahl Sektoren CH = max. Anzahl Spuren CF = Statusflagge AH = Statuscode	Statuscode in AH: s. Laufwerksroutine 1 (Diskettenstatus feststellen)
AT Festplattenparametertabelle initialisieren (zwei Plattenstationen)	13	AH = 09	CF = Statusflagge AH = Statuscode	Interrupt 41 zeigt auf die Tabelle für Laufwerk 0 Interrupt 46 zeigt auf die Tabelle für Laufwerk 1 Statuscode in AH: s. Laufwerksroutine 1 (Laufwerksstatus feststellen)
AT Lange Sektoren lesen	13	AH = 0A DL = Laufwerksnummer DH = Kopfnummer CH = Zylindernummer CL = Sektornummer ES:BX = Zeiger auf Puffer	CF = Statusflagge AH = Statuscode	Statuscode in AH: s. Laufwerksroutine 1 (Laufwerksstatus feststellen)

Routine	Interrupt (hex)	Register Eingabe	Ausgabe	Erläuterung
AT Lange Sektoren	13	AH = 0B DL = Laufwerksnummer DH = Kopfnummer CH = Sektornummer ES:BX = Zeiger auf Puffer	CF = Statusflagge AH = Statuscode	Statuscode in AH: s. Laufwerksroutine 1 (Laufwerksstatus feststellen)
AT Zylinder anfahren	13	AH = 0C DL = Laufwerksnummer DH = Kopfnummer CH = Sektornummer	CF = Statusflagge AH = Statuscode	Statuscode in AH: s. Laufwerksroutine 1 (Laufwerkstatus feststellen)
AT Alternativer Laufwerk-Reset	13	AH = 0D DL = Laufwerksnummer	CF = Statusflagge AH = Statuscode	Statuscode in AH: s. Laufwerksroutine 1 (Laufwerksstatus feststellen)
AT Test, ob Laufwerk bereit	13	AH = 10 DL = Laufwerksnummer	CF = Statusflagge AH = Statuscode	Statuscode in AH: s. Laufwerksroutine 1 (Laufwerksstatus feststellen)
AT Laufwerk neu kalibrieren	13	AH = 11 DL = Laufwerksnummer	CF = Statusflagge AH = Statuscode	Statuscode in AH: s. Laufwerksroutine 1 (Laufwerksstatus feststellen)
AT Controller-Diagnose	13	AH = 14	CF = Statusflagge AH = Statuscode	Statuscode in AH: s. Laufwerksroutine 1 (Laufwerksstatus feststellen)
AT Laufwerkstyp feststellen	13	AH = 15 DL = Laufwerksnummer	AH = Laufwerkstyp CX.DX = Anzahl der 512-byte-Sektoren, wenn AH = 3	Laufwerkstyp: AH = 0: kein Laufwerk AH = 1: Floppy ohne Möglichkeit, Diskettenwechsel zu erkennen AH = 2: Floppy mit Möglichkeit, Disketten zu erkennen AH = 3: Plattenstation
AT Diskettenwechsel erkennen	13	AH = 16	DL = Laufwerk mit Diskettenwechsel AH = Wechselstatus 00 = kein Wechsel 01 = Wechsel	
AT Disketten-laufwerk festlegen	13	AH = 17 AL = Laufwerkstyp		Laufwerkstypen in AL: AL = 00: kein Laufwerk angeschlossen AL = 01: normales Diskettenlaufwerk für 360-Kbyte-Disketten AL = 03: Diskettenlaufwerk für 1,2-Kbyte-Disketten

Routine	Interrupt (hex)	Register Eingabe	Ausgabe	Erläuterung
Routinen für seriellen Port				
Parameter für seriellen Port initialisieren	14	AH = 00 DX = Portnummer	AX = Portstatus	Statusbits: 00, 01 = Wortlänge 10 = 7-bit-Code; 11 = 8-bit-Code 02 = Stop-Bits; 0 = 1; 1 = 2 03, 04 = Parität: 00, 01 = keine, 01 = ungerade; 11 = gerade 05, 06, 07 = Übertragungsgeschwindigkeit in Baud: 000 = 110; 001 = 150; 010 = 600; 100 = 1,200; 101 = 2,400; 110 = 4,800; 111 = 9,600
Ein Zeichen senden	14	AH = 01 AL = Zeichen DX = Portnummer	AH = Übertragungsstatus AL = Modemstatus	Übertragungsstatus in AH: 00 = Daten bereit 01 = Überlauffehler 02 = Paritätsfehler 03 = Framing-Fehler 04 = Unterbrechungsfehler 05 = Übertragungspufferregister leer 06 = Übertragungsshiftregister leer 07 = Time-out-Fehler Modemstatus in AL: 00 = Delta Sendebereit (OCTS) 01 = Delta Datenendgerät bereit (DDSR) 02 = Abschließende Flanke (TERI) 03 = Delta Empfangsleitungssignal (DRLSO) 04 = Sendebereit (CTS) 05 = Datenendgerät bereit (DSR) 06 = Ringkomplement (RI) 07 = Empfangsleitungssignal (RLSO)
Ein Zeichen empfangen	14	AH = 02 DX = Portnummer	AH = Übertragungsstatus AL = Zeichen	Status: s. Portroutine 1 (Ein Zeichen senden)
	14	AH = 03	AX = Übertragungsstatus	Status: s. Portroutine 1 (Ein Zeichen senden)
Kassettenrecorderroutinen				
Motor einschalten	15	AH = 00	–	

Routine	Interrupt (hex)	Register Eingabe	Ausgabe	Erläuterung
Motor ausschalten	15	AH = 01	–	
Datenblöcke lesen	15	AH = 02 CX = Anzahl Bytes ES:BX = Zeiger auf Datenbereich	CF = Fehlerflagge DX = Anzahl gelesener Bytes ES:BX = Zeiger nach dem zuletzt gelesenen Byte	
Datenblöcke schreiben	15	AH = 03 CX = Anzahl Bytes ES:BX = Zeiger auf Datenbereich	ES:BX = Zeiger nach dem zuletzt geschriebenen Byte	
AT-Routinen				
AT Schnittstelle offen	15	AH = 80 BX = Gerätenummer CX = Prozeßart	–	
AT Schnittstelle geschlossen	15	AH = 81 BX = Gerätenummer CX = Prozeßart	–	
AT Programmende	15	AH = 82 BX = Gerätenummer	–	
AT Auf Ereignis warten	15	AH = 83 AL = Auswahl: 0 = Interval setzen 1 = abbrechen ES:BX = Zeiger auf Speicher des auftretenden Programms CX, DX = Wartezeit in Mikrosekunden	–	
AT Joystick	15	AH = 84 DX = 0 momentane Schaltstellungen abfragen	AL = Schaltstellungen	
AT Joystick	15	AH = 84 DX = 1 Werte abfragen	AX = A(x)Wert BX = A(y)Wert CX = B(y)Wert DX = B(y)Wert	
AT SysReg-Tastendruck	15	AH = 85 AL = 00 Drücken AL = 01 Unterbrechen	–	
AT Warten	15	AH = 86 CX, DX = Wartezeit in Mikrosekunden	–	

Routine	Interrupt (hex)	Register Eingabe	Ausgabe	Erläuterung
AT Block verlagern	15	AH = 87 CX, DX = Anzahl der zu verschiebenden Worte ES:SI = Zeiger auf Tabelle	–	
AT Erweiterte Speicherkapazität	15	AH = 88	–	
AT Auf virtuellen Speicher schalten	15	AH = 89	–	Vorsicht: s. BIOS-Listing
AT Gerät beschäftigt-Schleife	15	AH = 90 AL = Typencode	–	s. BIOS-Listing
AT Flagge setzen und Interrupt beenden	15	AH = 91 AL = Typencode	–	s. BIOS-Listing
Tastaturroutinen				
Zeichen des nächsten Tastendruckers holen	16	AH = 00	AH = Tastenauswahl-Code (Hilfs-Byte) AL = Zeichencode (Haupt-Byte)	
Meldung, ob Zeichen bereit	16	AH = 01	ZF = AH = Tastenauswahl Code (Hilfs-Byte) AL = Zeichencode (Haupt-Byte)	
Umschaltstatus feststellen	16	AH = 2	AL = Umschaltstatus	Umschaltstatus-Bits: Bit 0 = 1: rechte Shifttaste gedrückt Bit 1 = 1: linke Shifttaste gedrückt Bit 2 = 1: Ctrl gedrückt Bit 3 = 1: Alt gedrückt Bit 4 = 1: Scroll Lock aktiv Bit 5 = 1: Num Lock aktiv Bit 6 = 1: Caps Lock aktiv Bit 7 = 1: Einfügemodus
Druckerroutinen				
Ein Byte an den Drucker senden	17	AH = 00 AL = Zeichen DX = 00 ≙ LPT1: DX = 01 ≙ LTP2: DX = 02 ≙ LTP3:	AH = Statuscode	Status-Bits: 0 = Time out 1 = unbenutzt 2 = unbenutzt 3 = 1: E/A-Fehler 4 = 1: gewählt 5 = 1: Papier fehlt 6 = 1: bestätigt 7 = 1: nicht bestätigt

Routine	Interrupt (hex)	Register Eingabe	Ausgabe	Erläuterung
Drucker initialisieren	17	AH = 01 DX = 00 ≙ LPT1: DX = 01 ≙ LPT2: DX = 02 ≙ LPT3:	AH = Statuscode	Statuscode: s. Druckerroutine 0 (Ein Byte an den Drucker senden)
Druckerstatus feststellen	17	AH = 2 DX = 00 ≙ LPT1: DX = 01 ≙ LPT2: DX = 02 ≙ LPT3:	AH = Statuscode	Statuscode: s. Druckerroutine 0 (Ein Byte an den Drucker senden)
Diverse Routinen				
ROM-BASIC aktivieren	18	–	–	
Urladerstart-routine aktivieren	19	–	–	
Tageszeitroutinen				
Zählerstand lesen	1A	AH = 00	AL = Mitternachtssignal CX = Taktzähler (höherwertiger Teil) DX = Taktzähler (niederwertiger Teil)	
Zählerstand setzen	1A	AH = 01 CX = Taktzähler (höherwertiger Teil) DX = Taktzähler (niederwertiger Teil)	–	
AT Uhrzeit lesen	1A	AH = 02	CH = Stunden CL = Minuten DH = Sekunden	
AT Uhrzeit stellen	1A	AH = 03 CH = Stunden CL = Minuten	DH = Sekunden DL = 0 für Standardzeit	
AT Datum ablesen	1A	AH = 04	DL = Tag DH = Monat CL = Jahr CH = Jahrhundert (19 oder 20)	
AT Datum stellen	1A	AH = 05 DL = Tag DH = Monat CL = Jahr CH = Jahrhundert (19 oder 20)	–	
AT Alarm setzen	1A	AH = 06 CH = Stunden CL = Minuten DH = Sekunden	–	Adresse der Alarmroutine: Speicherstelle zu Interrupt 4A
AT Alarm rücksetzen	1A	AH = 07	–	

Kapitel 14
DOS-Grundlagen

14.1 Vor- und Nachteile bei der Verwendung von DOS-Routinen 222
14.1.1 Floppy- und Plattenroutinen 223
14.1.2 Bildschirmroutinen 223
14.2 Unterschiede der DOS-Versionen 224
14.3 Diskettenformate 225
14.4 Anmerkungen 225

Bevor wir uns in den Kapitel 15 bis 17 im Detail mit den DOS-Routinen beschäftigen, wollen wir nachfolgend die Merkmale, die allen DOS-Routinen gemeinsam sind, besprechen. Eine zusammenfassende Auflistung der DOS-Routinen finden Sie in Kapitel 18.
Die DOS-Routinen lassen sich in Interrupts und in Funktionsaufrufe unterteilen. Die Interrupts werden mit dem Befehl INT über einen Code aufgerufen. Die Funktionsaufrufe hingegen werden ähnlich den ROM-BIOS-Routinen angesprochen: durch einen Hauptinterrupt, hier Interrupt 33 (hex 21). Wie bei den BIOS-Routinen wird die Unterroutine durch eine Routinennummer in Register AH bestimmt. Die Aufteilung in Interrupts und Funktionsaufrufe resultiert aus Kompatibilitätserwägungen zu dem Betriebssystem CP/M und ist nur historisch zu verstehen.
Ein Vergleich zeigt, daß Interrupts nicht so effizient sind wie Funktionsaufrufe. Der Hauptvorteil der Funktionsaufrufe ist, daß sich problemlos Routinen hinzufügen lassen, ohne daß neue Interrupts geschaffen werden müssen. Es ist nicht verwunderlich, daß alle mit DOS Version 2 neu eingeführten Routinen als Funktionsaufrufe und nicht als Interrupts programmiert sind. Das gilt auch für DOS Version 3, allerdings wurde hier doch *ein* neuer Interrupt hinzugefügt.

14.1 Vor- und Nachteile bei der Verwendung von DOS-Routinen

Bei der Erstellung komplexer Programme tritt fast immer die Frage auf, ob DOS-Routinen verwendet werden sollen oder nicht. Grundsätzlich gilt auch hier der Rat, der Sie durch das gesamte Buch begleitet: Verwenden Sie stets die höchste Ebene, auf der Sie ein Programm erstellen können. Das wird im allgemeinen die Ebene der Programmiersprache sein. Greifen Sie nur in Ausnahmefällen direkt auf DOS oder BIOS Routinen zu. Die Programmierung auf der untersten Ebene, der Hardwareebene, sollten Sie vermeiden.
Fast immer gilt: Entweder nutzt ein Programm nur die Möglichkeiten der Programmiersprache oder aber es wickelt die *gesamte* Ein- und Ausgabe auf einer niederen Ebene ab. Im letzteren Fall stellt sich die Frage, ob BIOS- oder ob DOS-Routinen verwendet werden sollen. Die Entscheidung darüber sollte man nicht global, sondern in Abhängigkeit von den benötigten Funktionen fällen. Für Floppy- und Plattenoperationen ist DOS dem BIOS überlegen. Bei der Bedienung der Tastatur und anderer E/A-Geräte sind DOS und BIOS nahezu gleichwertig (je nach konkreter Anwendung). Im Bereich Bildschirmprogrammierung ist BIOS wesentlich leistungsfähiger als DOS.

14.1.1 Floppy- und Plattenroutinen

Wie Sie in den Kapiteln 16 und 17 erfahren werden, hängt der größte Teil der DOS-Aufgaben in irgendeiner Weise mit der Manipulation von Dateien auf Disketten bzw. Platten zusammen. Das bedeutet aber nicht, daß DOS ein in sich geschlossenes System zur Verwaltung einer Floppy oder Festplattenstation wäre. Ganz im Gegenteil! DOS ist viel eher eine Sammlung aufeinander abgestimmter Routinen, sozusagen ein "Werkzeugkasten".

14.1.2 Bildschirmroutinen

Es ist üblich, daß bei komplexer Software die Bildschirmausgabe auf einer relativ niedrigen Ebene abläuft. Häufig wird die gesamte Ausgabe durch direktes Schreiben in den Bildschirmspeicher abgewickelt. Andere Operationen wie z.B. das Bewegen des Cursors werden zumeist auf der nächsthöheren Ebene, dem ROM-BIOS, durchgeführt.
Als die ersten PCs auf den Markt kamen, war das Ausweichen auf untere Ebenen bei der Bildschirmprogrammierung notwendig, da DOS keine geeigneten Routinen anbot. Ab DOS 2.00 können aber mit dem ANSI-Treiberprogramm (ANSI.SYS) die meisten Bildschirmoperationen durchgeführt werden. Informationen über ANSI.SYS finden Sie in Anhang A. Das Programm erlaubt es, mit einem einheitlichen Befehlssatz praktisch alle Bildschirmfunktionen auszunutzen. Es ist aber problematisch, mit den ANSI-Treiberroutinen zu arbeiten, weil damit zwei Voraussetzungen verknüpft sind: Erstens müssen die Programme mit DOS 2.00 oder einer späteren DOS-Version arbeiten und zweitens muß DOS so konfiguriert werden, daß der ANSI-Treiber eingebunden ist. Viele Computerneulinge sind mit der Treibereinbindung überfordert und das allein ist schon ein gewichtiger Grund, auf DOS-Möglichkeiten, die den Treiber benötigen, zu verzichten.
Nun sind Sie vielleicht zu der Überzeugung gelangt, daß man generell auf den Einsatz der DOS-Bildschirmroutinen verzichten sollte. Die Problematik ist jedoch komplexer. Viele hochentwickelte Betriebssystemumgebungen, vor allem die Fenstersysteme, funktionieren nur zusammen mit Programmen, die den Bildschirm auf der DOS-Ebene verwalten. Ein direkter Hardwarezugriff führt in diesen Konfigurationen zu Problemen. Das ist ein bedeutsames Argument für die strikte Beschränkung auf die DOS-Bildschirmroutinen, soweit dies möglich ist.
Ein anderer Aspekt: Wenn ein Programm auf verschiedenen PC-Modellen mit unterschiedlicher Hardwareausstattung laufen soll, ist man als Programmierer praktisch gezwungen, auf der DOS-Ebene zu bleiben. Nur dadurch läßt sich eine ausreichende Kompatibilität herstellen. Andernfalls müßte man für jedes Modell eine Sonderanfertigung der Software vorneh-

men. Die Beschränkung auf DOS-Routinen ist ebenfalls ratsam, wenn ein Programm über mehrere Jahre hinweg eingesetzt werden soll. Zukünftige PC-Modelle dürften auf der DOS-Ebene aufwärtskompatibel bleiben, während auf der Hardwareebene inkompatible Veränderungen zu erwarten sind. Andererseits ist auf dem wichtigen Gebiet der Bildschirmprogrammierung die DOS-Ebene unzweifelhaft der BIOS-Ebene unterlegen. Die Entscheidung - DOS oder BIOS - ist daher oftmals schwierig.

14.2 Unterschiede der DOS-Versionen

DOS 3.10 ist die sechste offizielle DOS-Version. Jede neue Version enthielt Fehlernachbesserungen und Erweiterungen. Entscheidend für das Erscheinen neuer DOS-Versionen waren aber immer Hardwareentwicklungen, zumeist im Bereich Disketten-/Plattenstation.

Version	Datum	Hardwareänderung
1.00	4. 8.1981	Original-PC (Diskettenlaufwerk mit einseitiger Aufzeichnung)
1.10	7. 5.1982	Diskettenlaufwerk mit beidseitiger Aufzeichnung
2.00	8. 3.1983	XT (Festplattenstation)
2.10	20.10.1983	Tragbarer PC (Laufwerke in halbhoher Bauweise)
3.00	14. 8.1984	AT (1, 2 Mbyte Diskettenlaufwerk)
3.10	7. 3.1985	Netzwerkfähigkeit (netzwerkfähige Laufwerke)

Tabelle 14-1 Die sechs DOS-Versionen und die damit verbundene Änderung der Hardware

Bei allen Versionen außer 2.10 und 3.10 finden wir gravierende Änderungen in der Disketten /Plattenunterstützung (einschließlich neuer Diskettenformate). Die Hauptneuerung der Version 2.10 war nur eine Kleinigkeit, hatte aber ebenfalls mit Disketten zu tun: Die Zeit zur Kopfpositionierung wurde neu angepaßt, so daß die Unterschiede der halbhohen Laufwerke des tragbaren PC bezüglich der Datenaufzeichnung von den anderen Laufwerken toleriert wurden. Zudem wurden einige Fehler, die in Version 2.00 vorhanden waren, behoben. Mit der Version 3.10 wurden Netzwerkfunktionen eingeführt, die ursprünglich schon für Version 3.00 vorgesehen waren, dann aber nicht mehr rechtzeitig zum Freigabedatum fertiggestellt werden konnten. Nachfolgend ein kurzer Überblick über die Unterschiede der DOS-Versionen:

Version 1.00 arbeitete mit 8-Sektoren-Disketten. Alle grundlegenden DOS-Routinen waren hier schon zu finden.

Version 1.10 fügte die Unterstützung zweiseitiger Disketten hinzu. Die DOS-Routinen veränderten sich nicht.

Version 2.00 unterstützte zusätzlich Disketten- und Plattenaufzeichnungsformat mit 9 Sektoren. Die DOS-Routinen wurden umfassend er-

weitert (siehe Kapitel 17), außerdem wurde eine Cartridge-Unterstützung eingebaut.
Version 2.10 fügte weder neue Diskettenformate noch neue Routinen hinzu, berichtigte aber die Ausführungszeiten einiger Diskettenoperationen, um Kompatibilität zum tragbaren PC herzustellen.
Version 3.00 stellte die Möglichkeit zur Benutzung von 1,2 Mbyte Diskettenlaufwerken und weitere Festplattenformate bereit. Die Basis für netzwerkfähige Laufwerke ist hier schon zu finden.
Version 3.10 fügte die Netzwerkfähigkeit endgültig hinzu, die im wesentlichen auf dem Prinzip eines gemeinsamen Dateizugriffs beruht.

Hinweis: Jede neue DOS-Version ist bis auf wenige Details aufwärtskompatibel zu früheren Versionen. Abweichungen im Detail scheinen grundsätzlich unvermeidbar zu sein.

Für Programmierer stellt sich die Frage, auf welche DOS-Version man die eigene Software stützen soll. Derzeit scheint es am günstigsten zu sein, zu DOS 2.00 und 2.10 kompatibel zu sein. DOS 1 ist in den letzten Jahren immer mehr in den Hintergrund gedrängt worden, während DOS 3 *noch* keine weite Verbreitung gefunden hat.

14.3 Diskettenformate

Neben den Entscheidungen über die DOS-Version und die DOS-Routinen stellt sich die Frage nach dem Diskettenformat. Manche kommerziellen PC-Programme werden auf einseitigen Disketten mit acht Sektoren pro Spur angeboten. Der einzige, aber bedeutsame Vorteil: Diese Disketten können mit allen DOS-Versionen gelesen werden. Generell sind aber die einseitigen Diskettenformate mittlerweile durch die beidseitigen Formate abgelöst worden. Ab DOS 2.00 kann zudem mit einer Dichte von neun Sektoren pro Spur aufgezeichnet werden.
Ein Programm kann relativ sicher feststellen, unter welcher DOS-Version es läuft. Dazu ist nur der Funktionsaufruf 48 (hex 30) nötig. Sollten Sie nicht sicher sein, daß immer die richtige Version geladen ist, bauen Sie diese Sicherheitsfunktion in Ihre Programme ein. In Kapitel 17 finden Sie die notwendigen Erläuterungen.

14.4 Anmerkungen

Es gibt nur relativ wenige tiefgehende technische Informationen über DOS. Sowohl IBM als auch Microsoft halten sich in dieser Hinsicht recht bedeckt. Der Autor war daher beim Abfassen der folgenden Kapitel weitgehend auf eigene Erfahrungen angewiesen.

Kapitel 15
DOS-Interrupts

15.1 Die fünf Hauptinterrupts des DOS 228
15.1.1 Interrupt 32: Programm beenden 229
15.1.2 Interrupt 37 und 38: Diskette/Platte: Absolutes Lesen und Schreiben 229
15.1.3 Interrupt 39: Beenden und im Speicher verbleiben 232
15.1.4 Interrupt 47: Steuerung des Druckerspoolers 233
15.2 Die drei Adreßinterrupts des DOS 234
15.2.1 Interrupt 34: Endadresse 235
15.2.2 Interrupt 35: Unterbrechungsadresse 235
15.2.3 Interrupt 36: Fehlerbehandlungsadresse 236
15.3 Das Programmsegmentpräfix (PSP) 239
15.4 Die interne Struktur des PSP 240
15.5 Beispiel 244

In diesem Kapitel behandeln wir die DOS-Routinen, die von einem eigenen Interrupt aufgerufen werden. In den Kapiteln 16 und 17 werden die Funktionsaufrufe, die mit einer Routinennummer unter einem übergeordneten Interrupt angesprochen werden, erläutert.
Insgesamt gibt es neun Interrupts. Fünf davon, die Interrupts 32, 37 bis 39 und 47 (hex 20, 25 bis 27 und 2E) sind echte DOS-Interrupts, von denen jeder eine genau definierte Aufgabe erfüllt. Die anderen haben weniger fest umrissene Funktionen. Der vielleicht wichtigste Interrupt ist der Interrupt 33 (hex 21), der für die DOS-Funktionsaufrufe verwendet wird (siehe auch Kapitel 16 und 17). Die drei verbleibenden Interrupts, 34 bis 36 (hex 22 bis 24), enthalten segmentierte Adressen. Programme verändern die Adressen (vorzugsweise über den DOS-Funktionsaufruf 37), um auf bestimmte Routinen zu zeigen. Unter bestimmten Umständen ruft DOS diese Routinen über die Adreßinterrupts auf.

Interrupt Dez	Hex	Beschreibung
32	20	Programm beenden
33	21	Hauptinterrupt für Funktionsaufrufe
34	22	Endadresse
35	23	Unterbrechungsadresse
36	24	Fehlerbehandlungsadresse
37	25	Diskette/Platte: absolutes Lesen
38	26	Diskette/Platte: absolutes Schreiben
39	27	Beenden und im Speicher verbleiben
47	2F	Steuerung des Druckerspoolers (nur DOS 3)

Tabelle 15-1 Die neun Interruptroutinen des DOS

Hinweis: IBM geht offiziell davon aus, daß die einzelnen DOS-Interrupts nicht verwendet werden (außer natürlich für interne DOS-Zwecke). Konsequenterweise stehen unter dem Hauptinterrupt 33 (hex 21) Funktionsaufrufe zur Verfügung, die anstelle der DOS-Interrupts für die gleichen Funktionen benutzt werden können. Es besteht die Gefahr, daß zukünftige DOS-Versionen die nicht offiziell empfohlenen Routinen nicht mehr unterstützen, so daß Programme, die sich auf die Routinen beziehen, umgeschrieben werden müssen. Aus diesem Grund sollten Sie die Funktionsaufrufe den Interrupts vorziehen.

15.1 Die fünf Hauptinterrupts des DOS

Von den neun DOS-Interrupts sind fünf fest mit einer Routine verbunden, die eine fest umrissene Aufgabe erfüllt.

15.1.1 Interrupt 32 (hex 20): Programm beenden

Der Interrupt 32 wird zum Beenden eines Programmes verwendet, die Kontrolle geht an DOS zurück. Der Interrupt ist mit dem DOS-Funktionsaufruf 0 identisch (vergleichen Sie Kapitel 16.1.1). In jeder DOS-Version können beide Möglichkeiten gleichberechtigt benutzt werden, um ein Programm zu beenden.
Der Interrupt 32 schließt nicht automatisch die Dateien beim Verlassen des Programms. Vor dem Interrupt 32 sollten die DOS-Funktionsaufrufe 16 oder 62 aufgerufen werden, um Dateien, auf die schreibend zugegriffen wurde, zu schließen. Wenn eine Datei, die verändert wurde, nicht formal geschlossen wird, erscheint ihre neue Längenangabe nicht im Diskettenverzeichnis.
Um einen exakt definierten Einstieg in DOS zu gewährleisten, setzt Interrupt 32 die Adreßinterrupts 34, 35 und 36 auf ihre ursprünglichen Werte zurück. Das ist von essentieller Bedeutung, wenn das aufrufende Programm bereits von einem anderen Programm aus aufgerufen wurde. Nur so läßt sich verhindern, daß das Programm der obersten Programmebene an Routinen gelangt, die für die aufgerufenen Programme und deren Unterprogramme bestimmt sind. Die DOS-Funktion 75 (hex 4B) ist in diesem Zusammenhang auch erwähnenswert, lesen Sie hierzu auch in Kapitel 17 nach.

Hinweis: Bevor DOS ein Programm ausgeführt, wird ein Programmsegmentpräfix (PSP) an der Null-Offsetadresse des Codesegmentes erzeugt (das CS-Register zeigt auf den Null-Offset). Das PSP enthält unter anderem die Information, wo DOS weiterarbeiten soll, wenn das aufgerufene Programm beendet wird. Am Ende des Kapitels wird das PSP näher erläutert. Wenn der Interrupt 32 (Programm beenden) aufgerufen wird, richtet sich DOS nach dem Zeiger im CS-Register, der auf das PSP gerichtet sein muß. Wurde der Wert im CS-Register geändert, kann das zu Problemen führen.
Warnung: Wenn eine Routine über einen FAR-Befehl aufgerufen wird, ändert sich der Wert im CS-Register immer. Ein solches Programm sollte daher den Interrupt 32 nicht zum Beenden des Programmes verwenden.

15.1.2 Interrupt 37 und 38 (hex 25 und 26): Diskette/Platte: absolutes Lesen und Schreiben

Die Interrupts 37 und 38 dienen zum Lesen von bzw. Schreiben in Sektoren auf einer Diskette oder Festplatte. Dies sind die einzigen DOS-Routinen, die die logische Struktur des Datenträgers vollkommen ignorieren und sich direkt auf einzelne Sektoren beziehen. Dateien, Verzeichnisse und die FAT bleiben unbeachtet.
Die Interrupts 37 und 38 sind den entsprechenden BIOS-Routinen sehr ähnlich, mit der Ausnahme, daß die Sektoren mit einer anderen Numerie-

rungsmethode bezeichnet werden. Die ROM-BIOS-Routinen wählen die einzelnen Sektoren über ein dreidimensionales Koordinatensystem an (Spur/Zylinder, Seite/Kopf und Sektor), während die Interrupts 37 und 38 die Sektoren über eine sequentielle Numerierung bestimmen. Eine Erklärung der DOS-Sektornummerierung finden Sie in Kapitel 5.3
Mit den folgenden BASIC-Formeln lassen sich die dreidimensionalen BIOS-Koordinaten in die sequentielle DOS-Numerierung umwandeln:

```
DOS.SEKTOR.NUMMER = (BIOS.SEKTOR - 1) + BIOS.SEITE * SEKTOREN.PRO.SEITE
+ BIOS.SPUR * SEKTOREN.PRO.SEITE * SEITEN.PRO.LAUFWERK
```

Die Formeln für den umgekehrten Vorgang:

```
BIOS.SEKTOR = 1 + DOS.SEKTOR.NUMMER MOD SEKTOREN.PRO.SEITE
BIOS.SEITE = (DOS.SEKTOR.NUMMER \ SEKTOREN.PRO.SEITE) MOD SEITEN.PRO.LAUFWERK
BIOS.SPUR = DOS.SEKTOR.NUMMER \ (SEKTOREN.PRO.SEITE * SEITEN.PRO.LAUFWERK)
```

Hinweis: Der Wert SEKTOREN.PRO.SEITE beträgt für das beidseitige 9-Sektoren-Format (das geläufigste Format) 9, der Wert SEITEN.PRO.LAUFWERK beträgt 2. Beachten Sie, daß die Seiten und Spuren gegenüber BIOS anders numeriert werden: Seiten und Spuren beginnen bei 0, während die Nummer des ersten Sektors 1 ist.

Um einen ganzen Sektorblock anzusprechen, werden alle benötigten Parameter in verschiedene Register geladen: die Anzahl der Sektoren in das CX-Register, die Nummer des ersten Sektors in das DX-Register, die Speicheradresse für den Datentransfer in das Registerpaar DS:BX und das Laufwerk in das AL-Register (0 für Laufwerk A, 1 für Laufwerk B).
Während das ROM-BIOS mit physikalischen Laufwerken arbeitet, basieren die DOS-Routinen auf dem Konzept logischer Laufwerke. Dabei geht DOS von mindestens zwei logischen Laufwerken aus. Ist in Wirklichkeit gar kein Laufwerk B angeschlossen, nimmt DOS das eine vorhandene und behandelt es abwechselnd als A oder B, je nachdem, welches gerade benötigt wird. Diese logische Zuordnung kann mit dem DOS-Kommando ASSIGN geändert werden.
Die Ergebnisse der Interruptroutinen 37 und 38 werden durch eine Kombination der Carry-Flagge (CF), dem AL- und dem AH-Register ausgedrückt. CF ist 0, wenn kein Fehler aufgetreten ist. Für den Fall, daß ein Fehler entdeckt wurde (CF = 1), enthalten AL und AH den Fehlercode in zwei getrennten und etwas redundanten Kodierungen. Die Codes in AL sind die DOS-eigenen und basieren auf denen der Fehlerbehandlungsroutine, die über Interrupt 36 aufgerufen wird. Die Codes in AH stützen sich dagegen auf den Fehlercode des ROM-BIOS.
Interruptroutinen und Funktionsaufrufe stellen im allgemeinen den Stapel auf seinen ursprünglichen Zustand zurück, sobald sie mit der Ausführung ihrer Aufgabe fertig sind. Anders die Routinen, die mit den Interrupts 37 und 38 aufgerufen werden: Sie lassen ein Wort im Stapel zurück, das den Status des Flaggenregisters vor dem Aufruf der Routine widergibt. Das ist

Fehlercode Dez	Hex	Bedeutung
12	0C	Allgemeiner Fehler
11	0B	Lesefehler
10	0A	Schreibfehler
8	08	Gefragter Sektor ist nicht auffindbar
7	07	Unbekanntes Medium: Diskettenformat nicht erkannt
6	06	Anfahren der gewünschten Spur ist nicht möglich
4	04	Prüfsummenfehler bei CRC-Prüfung
2	02	Laufwerk nicht betriebsbereit (z. B. weil keine Diskette vorhanden oder die Laufwerksklappe nicht geschlossen ist)
1	01	Unbekannte Einheit: Ungültige Laufwerksnummer
0	00	Schreibschutzfehler: Versuch, auf schreibgeschützte Diskette zu schreiben

Tabelle 15-2 Die Fehlercodewerte und deren Bedeutung im AL-Register, die aus einem Fehler der Schreib- oder Leseroutinen (durch Interrupt 37 oder 38 aufgerufen) resultieren

Fehlercode Dez	Hex	Bedeutung
128	80	Time-out: Laufwerk reagiert nicht
64	40	Anfahren der gewünschten Spur ist nicht möglich
32	20	Fehlfunktion des Disketten-Controllers
16	10	Prüfsummenfehler bei CRC-Prüfung
8	08	DMA-Fehler
4	04	Gefragter Sektor ist nicht auffindbar
3	03	Schreibschutzfehler: Versuch, auf schreibgeschützte Diskette zu schreiben
2	02	Sektorkennung ungültig oder nicht auffindbar
0	00	Anderer Fehler

Tabelle 15-3 Die Fehlercodewerte im AH-Register, die von einem Fehler der Schreib- oder Leseroutinen (Interrupt 37 oder 38) herrühren und ihre Bedeutungen

sinnvoll, da die Interrupts 37 und 38 das Flaggenregister für die Ergebnismeldungen verändern. Andererseits kann ein Programm, das diese Routinen aufruft, Werte, die noch benötigt werden, mit PUSH auf den Stapel retten, wie das üblich ist. Ein besonderer Grund ist für diese Vorsichtsmaßnahme der Interrupts 37 und 38 nicht ersichtlich. In einem Programm, in dem die Interrupts 37 und 38 aufgerufen werden, sollten die zwei Flaggenstatus-Bytes aus dem Stapel entfernt (POP) und z.B. mit POP in das DX-Register oder (nach einem CF-Fehlertest) mit POPF in das Flaggenregister transferiert werden.

15.1.3 Interrupt 39 (hex 27): Beenden und im Speicher verbleiben

Interrupt 39 ist eine der interessantesten Routinen überhaupt, die von DOS zur Verfügung gestellt werden. Die Routine ist so wichtig, daß sie im Beispielprogramm am Ende des Kapitels erscheint.
Genau wie Interrupt 32 beendet auch Interrupt 39 ein Programm, ohne es - im Unterschied zu Interrupt 32 - aus dem Speicher zu löschen. Vielmehr verbleibt ein genau festgelegter Teil des Programmes im Speicher und der DOS-Eintrag, der auf den ersten freien Speicherbereich zeigt, wird auf die erste Segmentadresse direkt nach dem Programmrest gesetzt. Die Daten, die mittels des Interrupt 39 abgesondert sind, werden zu einer DOS-Erweiterung und können daher nicht durch andere Programme überschrieben werden. Nach offizieller IBM-Empfehlung sollte der DOS-Funktionsaufruf 49 (hex 32), der die gleiche Funktion wie Interrupt 32 aufweist, verwendet werden. In Kapitel 17 finden Sie hierzu weitere Informationen.
Interrupt 39 (oder der entsprechende Funktionsaufruf) wird von Programmen genutzt, die DOS-Erweiterungen installieren. Software zur Erweiterung der Tastaturfunktionen sprechen diesen Interrupt an. Häufig werden Interruptroutinen installiert, die bis zum Ausschalten des Computers im Speicher verbleiben sollen. Sehr oft ersetzen diese Routinen bereits vorhandene, um deren Funktion zu erweitern oder sogar völlig zu ändern. Diese Methode, Teile im Speicher zurückzulassen, ist nicht auf Programmroutinen begrenzt, ebensogut können Daten im Speicher verbleiben. Beispielsweise ist diese Technik zum Ablegen von Statusinformationen verschiedener Programme in einen gemeinsamen Speicherbereich brauchbar. Die Daten werden dadurch voneinander unabhängigen Programmen zugänglich.
Im allgemeinen verbleiben nur einige Routinen eines Programmes im Speicher, während der Rest gelöscht wird. Der Teil des Programmes, der erhalten bleiben soll, muß am Programmanfang stehen. Außerdem muß das Programm den Offset innerhalb des Codesegments auf das erste Byte nach dem zu erhaltenen Programmteil im DX-Register richten. Hierzu können Sie sich das Programmbeispiel in Kapitel 15.5 ansehen.
Alle Programme und Daten, die mit Interrupt 39 (bzw. DOS-Funktionsaufruf 49) im Speicher abgelegt werden, bleiben dort, solange DOS im Speicher existiert. Es ist durchaus nicht ungewöhnlich, daß verschiedene Programme Teile hinterlassen. Programme, die diese Technik verwenden, sind aber meist hochentwickelt und kompliziert; es ist daher ebenfalls nicht ungewöhnlich, daß gegenseitige Störungen auftreten. Um mit mehreren Programmteilen im Speicher erfolgreich arbeiten zu können, müssen die Teile manchmal in einer bestimmten Reihenfolge geladen werden. Die Reihenfolge läßt sich unter Umständen nur durch Experimentieren herausfinden. Wenn Sie ein Programm schreiben, das im Speicher verbleibt, sollten Sie keine allzu exotischen Programmiertechniken verwenden, um

wenigstens eine gewisse Kompatibilitätschance zu anderer Software zu erhalten.
Genau wie bei Interrupt 32 zum Beenden eines Programmes setzt DOS auch nach Ausführung von Interrupt 39 die Adreßvektoren für die Interrupts 34 bis 36 (hex 22 bis 24) zurück. Die logische Folge ist, daß sich über Interrupt 39 keine Interruptroutinen für die Adreßinterrupts installieren lassen. Obwohl das auf den ersten Blick eine Beschränkung darstellt, gibt es doch einen triftigen Grund dafür. Die Adreßinterrupts sind nämlich nicht zur allgemeinen Verwendung vorgesehen, sondern sollen nur von einzelnen Programmen gezielt eingesetzt werden. Wir werden dies an anderer Stelle noch genauer besprechen.

Hinweis: Bei einem EXE-Programm kann festgelegt werden, ob es ab der niedrigsten Speicherstelle nach oben hin (Standard) oder ab der höchsten Speicherstelle nach unten hin geladen werden soll. Lediglich Standard-EXE-Programme dürfen den Interrupt 39 verwenden, da er nur für Programme vorgesehen ist, die sich von unten nach oben aufbauen. In Kapitel 19.2.3 finden Sie Näheres über EXE-Programme.

15.1.4 Interrupt 47 (hex 2F): Steuerung des Druckerspoolers

Hinweis: Ein Großteil der Information, die Sie in diesem Kapitel finden, bezieht sich auf alle DOS-Versionen. Der Interrupt 47 ist hingegen nur in DOS 3.00 und späteren Versionen vorhanden.

Der neue DOS-Interrupt 47 (hex 2F), der mit der Version 3.00 eingeführt wurde, gibt Programmen Zugang zum DOS-Druckerspooler. Dieser Interrupt ist der normale Weg, auf einen Druckerspooler zuzugreifen, sei es der Standardspooler, der von DOS mit dem PRINT-Befehl bereitgestellt wird, oder ein anderer Druckerspooler.
Sechs Unterroutinen können über den Interrupt 47 aufgerufen werden, die Numerierung reicht von 0 bis 5. Der Aufruf erfolgt, indem die Routinennummer (Funktionscode) in das Register AH geladen und der Interrupt 47 ausgelöst wird.
Funktionscode 0 meldet, ob ein Druckerspooler installiert ist oder nicht. Das Ergebnis wird in das Register AL geschrieben. Der Wert 255 (hex FF) zeigt an, daß ein Spooler vorhanden ist und mutmaßlich verwendet werden kann. Der Wert 0 wird gemeldet, wenn keine Routine zur Bearbeitung des Interrupts zur Verfügung steht. Der Wert 1 zeigt an, daß kein Spooler installiert ist und auch keine Möglichkeit dazu besteht. Das bedeutet, daß der Interrupt 47 schon einem anderen Zweck dient und nicht für den Druckerspooler eingesetzt werden kann.
Funktionscode 1 wird verwendet, um dem Druckerspooler eine Datei, deren Inhalt ausgedruckt werden soll, zuzuweisen. Wir stellen den Zeiger im Registerpaar DS:DX auf einen 5-byte-Bereich, der *Zuweisungspaket*

genannt wird. In diesem Bereich werden alle für den Druck relevanten Informationen abgelegt. Das erste Byte ist ein Ebenencode, dessen Funktion dem Autor unbekannt ist, die anderen vier Bytes enthalten die segmentierte Adresse eines ASCIIZ-Strings (lesen Sie in Kapitel 17 nach), der den Pfadnamen der zu druckenden Datei enthält. Der Pfadname muß eine einzelne Datei sein. Die Verwendung der Dateigruppenzeichen "*" und "?" ist nicht erlaubt.
Die Datei wird an das Ende der Liste der zu druckenden Dateien gestellt. In der Reihenfolge der Zuweisung werden die Dateien ausgedruckt und aus der Warteschlange entfernt.
Funktionscode 2 entfernt einzelne Dateien aus der Spoolerwarteschlange. Das Registerpaar DS:DX zeigt auf den ASCIIZ-String, der die zu entfernende Datei definiert. Die globalen Zeichen "*" und "?" dürfen hier benutzt werden. Beachten Sie, daß das Registerpaar bei Funktion 2 direkt auf den Dateinamenstring zeigt und nicht, wie bei Funktion 1, auf das Zuweisungspaket, das auf den String verweist.
Funktionscode 3 leert die Spoolerwarteschlange vollständig. Bei beiden Funktionen (2 und 3) stoppt DOS einen gerade laufenden Ausdruck und gibt diesbezüglich eine kurze Meldung aus.
Funktionscode 4 gewährt Programmen Zugang zur Warteschlange des Spoolers, um sie einzusehen. Während des Zugriffs wird die Warteschlange *eingefroren*, um Änderungen in der Schlange zu unterbinden. Der Aufruf jedes anderen Funktionscodes des Interrupts 47 beendet den *Einfrierzustand*. Funktion 4 liefert als Ergebnis einen Zeiger in dem Registerpaar DS:SI, der auf die Namensliste der zu druckenden Dateien deutet. Die Einträge der Liste sind Strings mit einer festen Länge von 64 Bytes. Das Ende der Liste wird durch einen Eintrag angezeigt, der mit einem Null-Byte beginnt.
Funktionscode 5 hat keinen anderen Sinn, als den mit Funktion 4 hervorgerufenen *Einfrierzustand* aufzuheben. Die anderen Funktionen sind dazu aber ebenfalls geeignet.

15.2 Die drei Adreßinterrupts des DOS

DOS verwendet drei Interrupts, um außerordentliche Umstände zu verarbeiten, die Interrupts 34 bis 36 (hex 22 bis 24). Diese Umstände sind: das Ende eines Programmes, eine Unterbrechung (*Break*) per Tastatur (Ctrl-C oder Ctrl-Break) und ein sog. *kritischer Fehler* wie z.B. manche Laufwerksfehler. Programme können die Reaktionen auf die drei Fälle beeinflussen, indem sie die entsprechenden Interruptvektoren ändern. Die Vektoren dürfen auf jede beliebige Routine zeigen. Die drei Interrupts werden als *Adreßinterrupts* bezeichnet.
DOS enthält Standardwerte für jeden der drei Interrupts, die zu Beginn einer Programmabarbeitung gesichert und nach Beendigung wieder zu-

rückgesetzt werden. Das erlaubt es Programmen, die Adreßinterrupts frei zu verändern, ohne daß nachfolgende Programme oder die DOS-Kommandos selbst damit in Konflikt geraten.
Es ist möglich, die Standardwerte, die zu einem Programm im Programmsegmentpräfix (PSP) abgelegt sind, zu ändern (siehe hierzu auch Kapitel 15.3). In diesem Fall werden die modifizierten Werte zurückgespeichert, wenn das Programm beendet wird. Die wenigsten Programmierer bedienen sich dieser etwas "unsauberen" Technik, Sie sollten es auch nicht tun.

15.2.1 Interrupt 34 (hex 22): Endadresse

Die Adresse, die mit Interrupt 34 verbunden ist, legt fest, an welcher Speicherstelle nach Beendigung eines Programmes weitergearbeitet wird. Die Adresse wird auch in das PSP kopiert.
Meist wird die Routine verwendet, um die Kontrolle an den Befehlsinterpreter des DOS, COMMAND.COM, zurückzugeben, sobald ein Programm beendet ist. Während die zwei anderen Adreßinterrupts frei verändert werden können, so daß sie auf neue Routinen zeigen, hat der Endvektor im allgemeinen eine feststehende Funktion. Das läßt sich am besten an einem Beispiel erklären.
Ein Programm - nennen wir es *Prog1* - läßt DOS ein anderes Programm - *Prog2* - aufrufen. Ist *Prog2* beendet, wird an der Stelle fortgefahren, auf die der Interruptvektor 34 zeigt. Die Folge ist, daß DOS die Kontrolle übernimmt, wenn der ursprüngliche Wert in den Vektoradressen steht. Es kann sein, daß genau dies beabsichtigt war. Soll die Abarbeitung aber an *Prog1* zurückfallen, muß dieses den Vektor so verändern, daß er auf eine Stelle in *Prog1* zeigt. Die Veränderung muß stattfinden, bevor *Prog2* aufgerufen wird. Anschließend muß *Prog1* den Vektor wieder auf seinen Anfangswert zurücksetzen, damit nach Beendigung von *Prog1* die Kontrolle an DOS übergeben werden kann.
Im Gegensatz zu den zwei folgenden Adreßinterrupts wird der Interrupt 34 nicht erzeugt, sondern dient lediglich als Speicher einer segmentierten Adresse. Die Interrupts 29 bis 31, 68 und 73 (hex 1D bis 1F, 44 und 49) werden in ähnlicher Weise zum Ablegen von Adressen verwendet. Informationen hierzu finden Sie in Kapitel 3.2.1.
Das alles klingt reichlich exotisch. Arbeiten Sie aber mit dieser Möglichkeit nur, wenn Sie sie vollständig verstanden haben. Falls Ihnen das gelingt, werden Sie diese Dinge bestimmt besser begreifen, als man sie in einem Buch beschreiben kann.

15.2.2 Interrupt 35 (hex 23): Unterbrechungsadresse

Die Adresse von Interrupt 35 zeigt auf eine Routine, die immer dann aufgerufen wird, wenn DOS eine Unterbrechung seitens der Tastatur be-

antwortet. Die Unterbrechung wird auf der Standard-PC-Tastatur durch Ctrl-Break und auf jeder anderen Tastatur mit Ctrl-C (was aber nicht so weit verbreitet ist) ausgelöst.
DOS fragt nur während bestimmter Tastatur- und Bildschirmfunktionen ab, ob eine Unterbrechung vorliegt. Mit dem Kommando BREAK ON, das ab der DOS-Version 2.00 zur Verfügung steht, wird jederzeit auf eine Unterbrechung abgefragt und diese ggf. sofort bearbeitet.
Die normale DOS-Antwort auf einen Unterbrechungsinterrupt besteht darin, das laufende Programm oder die Batch-Kommandodatei, die gerade ausgeführt wird, zu beenden. Sie können jedoch jede andere Reaktion herbeiführen, indem Sie die Standardinterruptroutine durch eine eigene Routine ersetzen. Am Ende dieser Routine sollte die Kontrolle allerdings an DOS zurückgegeben werden, um einen Stapelüberlauf zu verhindern.
Die Möglichkeit, eine neue Routine zu installieren, wird zumeist genutzt, um entweder die Unterbrechung wirkungslos zu machen oder aber um den Abbruch einer Schleifenfunktion herbeizuführen. Im ersten Fall übergibt die Routine einfach die Kontrolle an das jeweilige Programm zurück. Im zweiten Fall setzt die Interruptroutine eine Flagge, die vom laufenden Programm, d.h. der Schleife, ständig abgefragt wird.
Es gibt für eine Unterbrechungsinterruptroutine zwei Möglichkeiten, die Kontrolle zurückzugeben. Die übliche Methode, die immer funktioniert, bedient sich des IRET-Befehls (*Interrupt Return*). Die Alternative dazu ist nur bei dem hier behandelten Interrupt möglich. Diese Interruptroutine hat nämlich die Möglichkeit, DOS mitzuteilen, ob ein Programm weiterlaufen oder beendet werden soll (in gleicher Weise funktioniert auch die normale Interruptbearbeitung, wenn kein Ersatz gegeben ist). Bei dieser Methode setzt die Routine die Carry-Flagge (CF) und endet mit einer FAR RET-Anweisung. Ist CF gleich 1, bricht DOS das Programm ab, bei CF gleich 0 wird das Programm fortgesetzt.

Hinweis: **Programme können eine Betätigung der Break-Taste simulieren, indem sie diesen Interrupt erzeugen.**

15.2.3 Interrupt 36 (hex 24): Fehlerbehandlungsadresse

Die Adresse des Interrupt 36 zeigt auf eine Routine, die aufgerufen wird, wenn ein *kritischer Fehler* (eine Situation, bei der keine Fortführung möglich ist) entdeckt wurde. Die meisten kritischen Fehler treten im Zusammenhang mit Diskettenlaufwerken auf. Aber auch andere Fehler werden gemeldet.
Wenn eine Fehlerbehandlungsroutine aufgerufen wird, stehen mehrere Informationsquellen über den Fehler selbst und über den Status vor dem Auftauchen des Fehlers zur Verfügung. Diese Quellen sind das Registerpaar BP:SI, der Stapel, das AH- und das DI-Register. Wir werden sie

Punkt für Punkt durchgehen, denn die Fehlerbehandlung ist insgesamt eine recht komplexe Angelegenheit.
Wenn Sie mit der DOS-Version 2.00 oder einer nachfolgenden arbeiten, zeigt das Registerpaar BP:SI auf einen Gerätekontrollblock. Ein Programm kann diesen Block heranziehen, um nähere Angaben über das Gerät (Diskettenlaufwerk, Drucker usw.) zu erhalten, das den Fehler verursacht hat.
Der Stapel enthält alle wichtigen Registerinhalte des Programmes, das die DOS-Funktion aufgerufen hat, die zu dem kritischen Fehler führte. Diese Information kann für ein Fehlerbehandlungsprogramm sehr nützlich sein, wenn es eng mit dem aktiven Programm zusammenarbeitet. Wenn wir annehmen, daß die Fehlerbehandlungsroutine mit traditionellen Methoden auf den Stapel zugreift, läßt sich der Stapel mit einer Offset-Adresse vom BP mit den zwei folgenden Befehlen lokalisieren.

```
PUSH  BP
MOV   BP,SP
```

Zuerst wird der ursprüngliche Basiszeiger (BP) gespeichert, dann wird BP gleich dem Wert des Stapelzeigers (SP) gesetzt. Diese Anweisungen finden Sie in Kapitel 8.4.4 wieder. Der Fehlerbearbeitungsroutine steht der Stapel zur Fehleranalyse zur Verfügung.

BP-Offsett	Stapelinhalt
0	BP, der auf den Stapel gelegt wurde
2	IP:CS der die Fehlerroutine aufrufenden DOS-Routine
6	Flaggen der die Fehlerbehandlungsroutine aufrufenden DOS-Routine
8	AX des die DOS-Routine aufrufenden Programmes
10	BX des die DOS-Routine aufrufenden Programmes
12	CX des die DOS-Routine aufrufenden Programmes
14	DX des die DOS-Routine aufrufenden Programmes
16	SI des die DOS-Routine aufrufenden Programmes
18	DI des die DOS-Routine aufrufenden Programmes
20	BP des die DOS-Routine aufrufenden Programmes
22	DS des die DOS-Routine aufrufenden Programmes
24	ES des die DOS-Routine aufrufenden Programmes
26	IP:CS des die DOS-Routine aufrufenden Programmes
30	Flaggen des die DOS-Routine aufrufenden Programmes

Tabelle 15-4 Der Stapelinhalt, wie er sich nach einer DOS-Routine, die zu einem kritischen Fehler führte, darstellt

Der kritische Fehler wird in erster Linie durch eine Kombination des höchstwertigen Bits von Register AH und des niederwertigen Bytes von Register DI (wahrlich eine seltsame Mischung) signalisiert. Ist das höchstwertige Bit des Registers AH gleich 0 (AH < 128), bezieht sich der Fehler immer auf eine Laufwerksoperation, andernfalls (AH > 127) kann es auch

Bit 2	1	0	Wert	Bedeutung
.	.	0	0	Lesefehler
.	.	1	1	Schreibfehler
0	0	.	0	Fehler bei DOS-Systemdateien
0	1	.	1	Fehler in der FAT
1	0	.	2	Fehler im Verzeichnis
1	1	.	1	Fehler im Datenbereich

Tabelle 15-5 Die Bitwerte und die dazugehörenden Fehler, die in den Bits 0 bis 2 des AH-Registers zu finden sind, nachdem Interrupt 36 aufgerufen wurde

eine andere Operation sein. Liegt ein Diskettenfehler vor (AH < 128), zeigt das AL-Register das betroffene Laufwerk an (0 ist Laufwerk A, 1 ist Laufwerk B usw.). Die Bits 0 bis 2 des AH-Registers beinhalten weitere Informationen über den Fehler.

Wenn der Wert in AH größer als 127 ist, liegt nicht notwendigerweise ein Laufwerksfehler vor, es kann aber einer sein. Einer der Diskettenfehler, der für gewöhnlich mit AH größer als 127 gemeldet wird, ist ein Fehler in der FAT der Diskette. Bei DOS Version 1 trifft das immer zu, ab Version 2.00 sollte die fehlerbehandelnde Routine den Gerätekontrollblock, auf den BP:SI zeigt, untersuchen. Wird als Gerät eine Diskettenstation gemeldet, liegt der Fehler in der FAT. Abgesehen von dieser Ausnahme zeigt das Register AH mit einem Wert größer als 127 eine Vielzahl von Fehlern für alle möglichen Geräte an, die keine Laufwerke sind. Dann muß man für nähere Informationen auf die Fehlercodes im niederwerti-

Fehlercode Dez	Hex	Bedeutung
12	0C	Allgemeiner Fehler
11	0B	Lesefehler
10	0A	Schreibfehler
9	09	Papierzufuhr gestört
8	08	Sektor nicht auffindbar
7	07	Unbekanntes Medium: Diskettenformat nicht erkennbar
6	06	Anfahren der Spur nicht möglich
5	05	Falsche Strukturlänge
4	04	Prüfsummenfehler bei CRC-Prüfung
3	03	Ungültiger Befehl an Controller gesendet
2	02	Laufwerk nicht betriebsbereit (z. B. weil keine Diskette vorhanden oder die Laufwerksklappe noch offen ist)
1	01	Ungültige Laufwerksnummer
0	00	Schreibschutzfehler: Versuch auf eine schreibgeschützte Diskette zu schreiben

Tabelle 15-6 Die Fehlerkodierung in Register DI, nachdem Interrupt 36 aufgerufen wurde

gen Byte von DI zurückgreifen, um das Problem exakt definieren zu können (das höherwertige Byte kann unbeachtet bleiben). Die Fehlercodes in DI sind im wesentlichen dieselben, die von den Interrupts 37 und 38 (hex 26 und 27) in das Register AL geschrieben werden.
Je nach Lage der Dinge braucht eine Fehlerbehandlungsroutine unter Umständen einige der DOS-Funktionsaufrufe, um dem Benutzer anzuzeigen, was passiert ist. Dazu können die einfachen Tastatur- und Bildschirmausgaberoutinen, Funktionsnummern 0 bis 12 (hex 0 bis C), verwendet werden. Routinen mit höheren Nummern, die meist Laufwerks- und andere Geräteoperationen anzeigen, sollten nicht benutzt werden, um die Komplexität in Grenzen zu halten.
Normalerweise wird die Kontrolle nach Beendigung einer Fehlerbearbeitungsroutine sofort an DOS zurückgegeben, das dann drei Dinge tun kann: den Fehler ignorieren, die Operation erneut versuchen oder das Programm beenden. Welche dieser drei Alternativen Sie für richtig halten, erfährt DOS durch den Wert im Register AL.

AL	DOS-Aktion
0	Fehler ignorieren und fortfahren
1	Operation erneut versuchen (das Problem mag zwischenzeitlich gelöst sein)
2	Programm beenden (DOS gibt den Interrupt 35 (hex 23) aus, erzeugt also eine Unterbrechung)

Tabelle 15-7 Die Werte in Register AL, die DOS anzeigen, was nach einer Fehlerbehandlungsroutine zu tun ist

Hinweis: **Da der Einleitungsprozeß zu Interrupt 36 sehr komplex ist, ist es nicht ratsam, einen kritischen Fehler durch Aufruf dieses Interrupts zu erzeugen.**

15.3 Das Programmsegmentpräfix (PSP)

Wenn DOS ein Programm lädt, wird zunächst ein Speicherbereich für das Programm angelegt, der *Codesegment* oder *Programmsegment* genannt wird. Dann reserviert DOS einen Block mit 256 (hex 100) Bytes am Anfang des Codesegmentes, das *Programmsegmentpräfix* (PSP). Das PSP ist ein Kontrollblock mit den wichtigsten Informationen über das Programm, das zumeist unmittelbar auf das Programmsegmentpräfix ab der Offset-Adresse hex 100 folgt.
Die Informationen im PSP sind notwendig, um ein Programm unter DOS ablaufen zu lassen. Das PSP ist Teil jedes DOS-Programmes ungeachtet der Programmiersprache, in der das Programm verfaßt ist. Jedes Programm, das das CS-Register abfragen kann, ist auch in der Lage, auf das Programmsegmentpräfix zuzugreifen. Die höheren Programmiersprachen

bringen allerdings überwiegend eine eigene Speicher- und Datenverwaltung mit, so daß sie auf das PSP nicht angewiesen sind. Für die Assemblerprogrammierung ist das Programmsegmentpräfix aber eine wichtige Informationsquelle.
Bevor wir die einzelnen Elemente des PSP besprechen, wollen wir noch die Verbindung zwischen PSP und zugehörigem Programm erläutern.
Das PSP beginnt bei der Offsetadresse 0 im Codesegment. Das eigentliche Programm folgt gewöhnlich direkt im Anschluß bei Offset hex 100, kann aber auch weiter hinten im Speicher liegen. Sobald das Programm abgearbeitet wird, werden bestimmte Register so gesetzt, daß sie auf das PSP verweisen. Bei einem einfachen .COM-Format-Programm zeigen alle Segmentregister auf den Anfang des PSP und das eigentliche Programm beginnt bei Offset hex 100. Bei dem komplexeren .EXE-Format, das die DOS-Operation LINK verwendet, verweisen nur die Register DS und ES auf das PSP. Das LINK-Programm sieht die Register CS, IP, SS und SP ein und kann folglich mit dem Registerpaar CS:IP den Beginn des Programmes auf eine andere Speicheradresse als hex 100 legen.
Gleichgültig, wo das Programm anfängt, bleibt die Verbindung zwischen dem PSP und dem Programm stets dieselbe. Entscheidend ist, daß ein Programm zu Beginn der Ausführung die Möglichkeit hat, durch eines der Segmentregister auf das PSP zuzugreifen. Wenn der Zeiger auf das PSP gespeichert wird, kann das Programm unabhängig von Änderungen in den Segmentregistern zu jedem beliebigen Zeitpunkt auf das PSP zugreifen.

15.4 Die interne Struktur des PSP

Der Inhalt des PSP stellt eine recht "bunte Mischung" verschiedenartiger Daten dar. Das ist nur historisch zu verstehen. Einerseits ist das heutige DOS eine Fortentwicklung des einst führenden Betriebssystems CP/M, andererseits geht die DOS-Entwicklung seit Jahren in Richtung UNIX. Das Ergebnis sehen Sie im PSP, das Elemente beider Betriebssystemkonzeptionen enthält. Lassen Sie sich dadurch nicht irritieren, wir werden das PSP im Detail besprechen.
Feld 1 enthält die Bytes hex CD und 20, die Anweisung zum Aufruf von Interrupt 32 (hex 20). Wie Sie wissen (Interrupt 32 wurde in diesem Kapitel behandelt), ist die Verwendung des Interrupt 32 eine gängige Methode, um einen Programmablauf zu beenden. Der Befehl steht am Beginn des PSP, Offset-Adresse 0, damit ein Programm direkt über das CS-Register an das PSP springen und die Abarbeitung abbrechen kann. Der Weg über den Interrupt oder den Funktionsaufruf ist allerdings vorzuziehen. Vermutlich wurde die PSP-Möglichkeit als "Notschalter" konzipiert. Wenn ein aus mehreren Teile zusammengebundenes Programm eine unerfüllbare Referenz enthält, kann die Programmabarbeitung mit einem Sprung zu Offset 0 gestoppt werden.

Feld	Offset Dez	Hex	Länge	Beschreibung
1	0	0	2	INT 32 Anweisung
2	2	2	2	Speichergröße (in Segmenten)
3	4	4	1	Reserviert; normalerweise 0
4	5	5	5	DOS-Funktionsaufruf – Dispatcher
5	10	A	4	Endvektor
6	14	E	4	Unterbrechungsvektor
7	18	12	4	Fehlerbehandlungsvektor
8	22	16	22	DOS-interne Verwendung
9	44	2C	2	Umgebungsstringzeiger
10	46	2E	34	DOS-Arbeitsbereich
11	80	50	3	INT 33, RETF Anweisungen
12	83	53	2	Reserviert
13	85	55	7	FCB 1 Erweiterung
14	92	5C	9	FCB 1
15	101	65	7	FCB 2 Erweiterung
16	108	6C	20	FCB 2
17	128	80	1	Parameterlänge
18	129	81	127	Parameter
19	128	80	128	Diskettentransferbereich (DTA)

Tabelle 15-8 Die Teile des PSP (Programmsegmentspräfix)

Feld 2 gibt Auskunft über den verfügbaren Speicher, indem die Segmentadresse des Speicherendes von DOS angezeigt wird. Sie brauchen diesen Wert nur mit 16 zu multiplizieren, um den zur Verfügung stehenden Speicherplatz in Bytes zu erhalten. Der DOS-Befehl CHKDSK meldet denselben Wert. Beachten Sie, daß der angezeigte Wert nicht unbedingt der tatsächlich physikalisch vorhandenen Speicherkapazität entsprechen muß. Viele RAM-Disks siedeln sich im oberen Speicherbereich an und setzen den DOS-Eintrag auf den Wert, bei dem der normale Speicherbereich endet. In einem Programm, das den gesamten Speicher nutzt, sollte die Kapazität stets im PSP abgefragt werden. Der Einsatz der BIOS-Interruptroutine 18 ist zu diesem Zweck nicht zu empfehlen.
In der normalen DOS-Umgebung steht der gesamte Speicherplatz einem einzelnen ablaufenden Programm zur Verfügung. In Fenster- oder Multitasking-Systemen, bei denen sich verschiedene Programme den Speicher teilen müssen, kann ein Programm einen Teil des Speichers, den es benötigt, für sich reservieren und den Rest mit der DOS-Funktion 74 (hex 4A), dem SETBLOCK-Befehl, freigeben. Diese Funktion erfüllt zwar alle DOS-Konventionen hinsichtlich der Speicherverwaltung, zusammen mit verschiedenen Fenstersystemen treten aber dennoch Probleme auf. In diesen Fällen kann das Feld 4 als Indikator für den freien Speicherbereich verwendet werden.

Feld 4 hat eine größere Bedeutung, als es auf den ersten Blick scheint. Generell gesehen ist das Feld ein verlängerter Arm zum DOS-Funktions-Dispatcher, der zum Aufrufen von DOS-Funktionen verwendet werden kann (was aber nach Möglichkeit vermieden werden sollte). Feld 4 dient aber eigentlich einer indirekten Angabe, ob ein Programm weniger als 64 Kbyte Speicherplatz zur Verfügung hat. Bei einem Aufruf enthält die Anweisung die Adresse der DOS-Funktions-Dispatcher-Routine; als intersegmentierter Aufruf liegt die Adresse im segmentierten Format vor. Die segmentierte Adresse wird so unterteilt, daß sie zwei Zwecken gerecht wird: Sie verweist auf den DOS-Funktions-Dispatcher und gibt über den Offset-Teil an, wieviel Platz des Codesegmentes noch zur freien Verfügung steht (bis zu hex FFF0, 16 Bytes weniger als 64 Kbyte). Der Offset-Teil steht bei Offset 6 im PSP, unmittelbar hinter dem Befehls-Opcode bei Offset 5.
Durch Feld 5 läßt sich leicht feststellen, wieviel Speicherplatz zur Verfügung steht, falls weniger als 64 Kbyte vorhanden sind. Die Methode sollte unter allen Fenster- und Multitasking-Systemen funktionieren. Sind mehr als 64 Kbyte Speicher, läßt sich die freie Kapazität aus Feld 2 ermitteln. Wie schon erwähnt kann diese Angabe aber unter Umständen falsch sein, wenn mit Fenster- oder Multitasking-Systemen gearbeitet wird.
Die Felder 5, 6 und 7 enthalten die Werte, die im Normalfall den drei Adreßinterrupts zugeordnet werden. Die Adressen werden zu Beginn der Programmabarbeitung gespeichert und bei Beendigung wieder in den Feldern 5, 6 und 7 abgelegt. Das ermöglicht es den Programmen, verschiedene Routinenvektoren während des Programmablaufes zu verwenden, ohne daß nachfolgende Programme oder gar DOS selbst davon beeinflußt werden. Falls keine neuen Vektoren bereitgestellt werden, verwendet DOS die im PSP gespeicherten Werte, die auf die zugeordneten Routinen zeigen. Wenn wir mit den im PSP gespeicherten Adressen arbeiten, können wir die Werte, die später zurückgespeichert werden, verändern (und damit natürlich auch die zugeordneten Routinen). Dadurch wird DOS verändert. Es gibt für gewöhnlich allerdings keinen Grund, einen dieser Werte zu lesen oder gar zu ändern.
Feld 9 enthält eine Segmentadresse, die auf die sog. *Umgebungsstrings* zeigt, die mit DOS 2.00 eingeführt wurden. Die *Umgebung* beginnt im Segment bei Offset 0.

Hinweis: Um eine Verwirrung in der Terminologie zu vermeiden, sei eine kurze Erklärung zum Wort Umgebung gegeben. In Zusammenhang mit dem PSP bezeichnet der Begriff Umgebung einen Stringblock (das wird gleich noch beschrieben), der von DOS verwendet wird, um bestimmte Informationen von Programm zu Programm zu übertragen. Das Wort Umgebung bezieht sich in diesem Buch ansonsten in recht freier Weise auf die Betriebsbedingungen, unter denen ein Programm abläuft.

Die DOS-Umgebung besteht aus einem Block mit ASCIIZ-Strings. Das sind Strings, die aus ASCII-Zeichen bestehen und mit dem Zeichen CHR$(0) enden. Sie können eine Vielzahl verschiedenartiger Informationen enthalten. Das Ende jeder Umgebungsfestlegung wird durch CHR$(0) markiert. Beginnt eine Umgebungsfestlegung mit CHR$(0), sind keine Strings darin enthalten.
Es ist Konvention, daß der String in der Form NAME=wert vorliegt. NAME muß in Großbuchstaben geschrieben sein und darf jede vernünftige Länge besitzen, jeder gültige Wert ist erlaubt. DOS setzt eine Umgebung für den Kommandoprozessor, die an jedes aufrufende Programm übergeben wird. Diese Umgebung enthält mindestens den Namen COMSPEC (wird von DOS verwendet, um die COMMAND.COM-Datei auf der Diskette zu finden), meist sind noch weitere Namen wie PATH oder SWITCHAR vorhanden. Mit dem DOS-Kommando SET lassen sich Umgebungsstrings hinzufügen, verändern oder löschen.
Feld 11 enthält zwei Anweisungen, die eine DOS-Funktion aufrufen (Interrupt 33, hex 21) und die Rückkehr zum aufrufenden Programm veranlassen (RETF oder FAR-Rücksprung). Dadurch ist es möglich, DOS-Funktionen halbwegs indirekt aufzurufen. Dazu werden alle Parameter für einen normalen Funktionsaufruf gesetzt (Spezifikation der Routine in Register AH usw.), der Aufruf erfolgt jedoch nicht über Interrupt 33 (eine 1-byte-Anweisung), sondern über einen FAR-Aufruf zur Offset-Adresse hex 50 im PSP (eine 5-byte-Anweisung).
Sie werden vielleicht vermuten, daß auch das ein Relikt aus der Vergangenheit ist, z.B. aufgrund von Kompatibilitätserwägungen mit CP/M. Erstaunlicherweise wurde Feld 11 aber erst mit der DOS-Version 2.00 eingeführt. Daraus läßt sich schließen, daß diese Art, eine DOS-Funktion aufzurufen, durchaus zukunftsorientiert ist.
Die Felder 13, 14, 15 und 16 unterstützen die herkömmliche Methode der Dateibehandlung unter Verwendung der Dateikontrollblöcke (*File Control Block* oder FCB). FCBs können in jeder DOS-Version verwendet werden, aber ab Version 2.00 kann man davon nur noch abraten, da die Dateiein-/ausgabe hier von Dateiunterstützungsroutinen übernommen wird. Mehr über FCBs finden Sie in Kapitel 16.2, mehr über Dateinummern in 17.
Die Felder 13, 14 und 25 wurden in das PSP aufgenommen, um die Erstellung von Programmen, die ein oder zwei Dateinamen als Parameter erhalten, zu erleichtern. Die dahintersteckende Idee ist, daß DOS die notwendigen FCBs aus den ersten zwei Programmparametern (die dem Programmnamen in der Kommandozeile unmittelbar folgen) konstruiert. Wenn ein Programm einen oder beide FCBs benötigt, kann es sie öffnen und verwenden, ohne die Befehlsparameter zu dekodieren und die FCBs selbst erstellen zu müssen.
Der Einsatz des PSP zum Aufbau von FCBs ist nicht unproblematisch. Erstens überlappen sich die beiden FCBs in der ursprünglichen Lage. Wenn Sie nur einen der beiden FCBs brauchen - gut! Benötigen Sie aber beide, sollten Sie einen oder sogar beide vor der Verwendung zu einem

anderen Platz schaffen. Die FCBs können auch FCB-Erweiterungen aufrufen, eine Tatsache, die in vielen DOS-Erläuterungen im Zusammenhang mit dem PSP übersehen wird.
Die Felder 17 und 18 ermöglichen Programmen den Zugriff auf die Parameter in der Kommandozeile. In Feld 17 steht die Gesamtlänge der Parameterzeichenkette (ein String bestehend aus 0 bis 127 Zeichen), Feld 18 enthält die Inhalte.
Bei der Parameterzeichenkette gibt es einige Besonderheiten zu beachten. Der String enthält nicht den Programmnamen, sondern beginnt mit dem ersten Zeichen unmittelbar hinter den Namen, das ist normalerweise ein Leerzeichen. Trennzeichen wie Leerzeichen oder Kommata bleiben unverändert im String enthalten und müssen bei einer Auswertung des Strings berücksichtigt werden. Ab DOS 2.00 werden alle Ein-/Ausgabeumleitungen wie <INPUT oder >OUTPUT von DOS aus der Kommandozeile herausgenommen. Es wird eine neue Kommandozeile ohne diese Elemente aufgebaut. Die Feststellung des Programmnamens und der Ein-/Ausgabeumleitungen ist also mit den Feldern 17 und 18 nicht möglich.

Hinweis: Die Felder 17 und 18 überlappen mit Feld 19 des PSP. Daher sollten Programme die Parameter lesen, bevor andere Operationen durchgeführt und die Parameter unter Umständen überschrieben werden.

Feld 19 ist der Standard-Diskettentransferbereich (DTA). Der DTA-Puffer hat eine Länge von 128 Bytes und beginnt bei PSP-Offset hex 80. Sobald eine DOS-Routine aufgerufen wird, die den DTA benötigt, wird er eingerichtet, sofern noch nicht geschehen. In den Kapiteln 16 und 17 finden Sie Hinweise zur Manipulation des Diskettentransferbereiches.
Obwohl das PSP üblicherweise als eine Einheit angesehen wird, ist es sinnvoll, zwei Teile zu unterscheiden. Der erste Teil besteht aus den ersten 92 Bytes des 256-Byte-PSP (Felder 1 bis 13, Offsets hex 0 bis 5B). Wenn Sie mit dem PSP arbeiten wollen, sollten Sie sich auf den zweiten Teil, also die Felder 14 bis 19 (Offsets hex 5C bis FF), beschränken. Das folgende Programmbeispiel zeigt Ihnen, wie Sie den ersten Teil des PSP unberührt lassen, während der zweite vom aktuellen Programm verwendet wird.

15.5 Beispiel

Unser Beispiel zeigt ein Programm, das den Interrupt *Beenden und im Speicher verbleiben*, Nummer 39, verwendet. Die meisten Beispiele in diesem Buch sind Schnittstellen zwischen einer höheren Programmiersprache und den Routinen des DOS oder ROM-BIOS. Das Beispielprogramm dieses Kapitels ist hingegen eine eigenständige Assemblerroutine, die über Interrupt 39 einen Teil der Routine im Speicher installiert.

Wie Sie gelernt haben, wird immer dann ein PSP von 256 Byte Länge eingerichtet, wenn ein Programm zur Ausführung geladen wird. Direkt an das PSP schließt sich das Programm an. Das CS-Register wird auf das PSP gesetzt, so daß ein Programm leicht auf die Informationen des PSP zugreifen kann. Da außerdem der IP-Offset zum Programmanfang stets hex 100 ist, kann über CS der Offset des Programmteils, der im Speicher fest installiert werden soll, errechnet werden. Die Formel dazu lautet:

```
SPEICHER_OFFSET  EQU  SPEICHER_ANFANG - ANFANG + 100H
```

Es ist Platzverschwendung, das PSP einer Routine, die im Speicher bleiben soll, stehen zu lassen. Andererseits wird das PSP benötigt, damit DOS ein Programm beenden kann. Damit der PSP-Bereich nicht sinnlos brachliegt, kann man zwei Dinge tun. Erstens: Benötigt das im Speicher verbleibende Programm einen Datenbereich, wenn es als Interruptroutine aktiviert wird, kann der PSP-Speicherplatz verwendet werden, der für DOS zur Programmbeendigung nicht erforderlich ist. Die zweite Möglichkeit besteht darin, daß ein Teil des Programmes selbst in das PSP geschrieben wird. Im folgenden Beispielprogramm wird das PSP bis auf die ersten 92 (hex 5C) Bytes völlig neu belegt. Der vordere PSP-Bereich sollte - es sei nochmals darauf hingewiesen - stets unverändert bleiben, um Komplikationen mit DOS zu vermeiden.
Wenn Sie mit dem Beispielprogramm experimentieren und anschließend die DOS-Funktion CHKDSK zum Testen des Speicherverbrauchs aufrufen, werden Sie feststellen, daß DOS immer mindestens 16 Bytes für interne Verwaltungszwecke belegt. Schon für sehr kleine Operationen können mehrere hundert Bytes mehr benötigt werden.
Hier nun das Programm:

```
; Beispiel zu DOS-Interrupt 39 (hex 27): Beenden und im Speicher verbleiben
PROGRAM  SEGMENT PARA PUBLIC 'CODE'
ASSUME   CS:PROGRAM
TBSR     PROC
            ANFANG:
;Sprung zur Initialisierungsroutine
            JMP     INITIALISIERUNG
            SPEICHER_ANFANG:
;hier steht der Teil, der im Speicher installiert werden soll
            SPEICHER_ENDE
            SPEICHER_LAENGE  EQU  SPEICHER_ENDE  - SPEICHER_ANFANG
            SPEICHER_OFFSET  EQU  SPEICHER_ANFANG - ANFANG + 100H
            PSP_LAENGE       EQU  92  ;der zu erhaltende Teil
;die folgenden Initialisierungsbefehle werden durch
;Interrupt 39 gelöscht
            INITIALISIERUNG:
```

```
;hier stehen Befehle zur Initialisierung des Programms, falls ;nötig
;es folgt die Routine, die den Programmteil, der im Speicher ;erbleiben soll,
größtenteils in den das PSP verschiebt
        PUSH    CS                  ;Zielsegment-...
        POP     ES                  ;...register setzen
        PUSH    CS                  ;Quellsegment-...
        POP     DS                  ;...register setzen
        MOV     DI,PSP_LAENGE       ;Ziel-Offset setzen
        MOV     SI,SPEICHER_OFFSET  ;Quell-Offset setzen
        MOV     CX,SPEICHER_LAENGE  ;zu transferierende Länge
        CLD                         ;setzt "vorwärts" für
                                    ;wiederholte MOV-Anweisung
        REPMOVSB                    ;transferiert die Programm-
                                    ;Bytes in den PSP-Bereich
;nun kann das Programm beendet werden; die erste Speicherstelle
;nach dem im Speicher verbleibenden Programm wird als erste freie
;Speicherstelle ausgewiesen
        MOV     DX,PSP_LAENGE + SPEICHER_LAENGE
        INT     27H
TBSR    ENDP
PROGRAMM ENDS
        END
```

Kapitel 16
Traditionelle DOS-Funktionen

16.1 Die traditionellen DOS-Funktionen 249
16.1.1 Funktion 0: Programm beenden 251
16.1.2 Funktion 1: Tastatureingabe mit Echo 251
16.1.3 Funktion 2: Bildschirmausgabe 252
16.1.4 Funktion 3: Serielle Eingabe 252
16.1.5 Funktion 4: Serielle Ausgabe 253
16.1.6 Funktion 5: Druckerausgabe 253
16.1.7 Funktion 6: Direkte Tastatur/Bildschirm-Ein-/Ausgabe 253
16.1.8 Funktion 7: Direkte Tastatureingabe ohne Echo 254
16.1.9 Funktion 8: Tastatureingabe ohne Echo 254
16.1.10 Funktion 9: Zeichenkette (String) darstellen 254
16.1.11 Funktion 10 (hex A): Gepufferte Tastatureingabe 255
16.1.12 Funktion 11 (hex B): Tastatureingabestatus prüfen 256
16.1.13 Funktion 12 (hex C): Tastaturpuffer löschen und Funktion ausführen 256
16.1.14 Funktion 13: Laufwerks-Reset 257
16.1.15 Funktion 14: Standardlaufwerk bestimmen 257
16.1.16 Funktion 15: Datei öffnen 258
16.1.17 Funktion 16: Datei schließen 258
16.1.18 Funktion 17: Nach erster passender Datei suchen 259
16.1.19 Funktion 18: Nach nächster passender Datei suchen 260
16.1.20 Funktion 19: Datei löschen 260
16.1.21 Funktion 20: Datensatz sequentiell lesen 260
16.1.22 Funktion 21: Datensatz sequentiell schreiben 261
16.1.23 Funktion 22: Datei anlegen 261
16.1.24 Funktion 23: Datei umbenennen 262
16.1.25 Funktion 24: Wird intern von DOS benutzt 262
16.1.26 Funktion 25: Standardlaufwerk feststellen 262
16.1.27 Funktion 26: Diskettentransferbereich (DTA) festlegen 263
16.1.28 Funktion 27: FAT-Informationen des Standardlaufwerkes lesen 263
16.1.29 Funktion 28: FAT-Informationen eines beliebigen Laufwerkes lesen 264

16.1.30 Funktion 33: Datensatz wahlfrei lesen 264
16.1.31 Funktion 34: Datensatz wahlfrei schreiben 265
16.1.32 Funktion 35: Dateilänge feststellen 265
16.1.33 Funktion 36: Feld für wahlfreien Zugriff setzen 265
16.1.34 Funktion 37: Interruptvektor setzen 266
16.1.35 Funktion 38: Programmsegment anlegen 266
16.1.36 Funktion 39: Datensätze wahlfrei lesen 266
16.1.37 Funktion 40: Datensätze wahlfrei schreiben 267
16.1.38 Funktion 41: Dateiname durchsuchen 267
16.1.39 Funktion 42: Datum ablesen 269
16.1.40 Funktion 43: Datum stellen 269
16.1.41 Funktion 44: Tageszeit ablesen 269
16.1.42 Funktion 45: Tageszeit stellen 270
16.1.43 Funktion 46: Diskettenschreibverifikation setzen 270
16.2 Der Dateikontrollblock (FCB) 270
16.3 Beispiel 274

In diesem Kapitel werden wir die Funktionen behandeln, die in allen DOS-Versionen enthalten sind. In der DOS-Terminologie werden diese alten (oder allgemeinen) Routinen häufig als *traditionelle Funktionen* bezeichnet. Die neuen Routinen, die Sie in Kapitel 17 erklärt finden und die mit der DOS-Version 2.00 eingeführt wurden, heißen *erweiterte Funktionen.*

16.1 Die traditionellen DOS-Funktionen

Alle DOS-Funktionsaufrufe werden durch Interrupt 33 (hex 21) angewählt. Welche Routine aufgerufen wird, bestimmt die Routinennummer in Register AH.
Bei der Einteilung der traditionellen DOS-Funktionen in logische Gruppen wird zum Teil von anderen Zuordnungskriterien als im DOS-Handbuch ausgegangen.

Funktion Dez	Hex	Gruppe
0	00	Keine Ein- oder Ausgabe
1–12	01–0C	Ein- und Ausgabe von Zeichen
13–36	0D–24	Dateimanagement
37–38	25–26	Ebenfalls keine Ein- oder Ausgabe
39–41	27–29	Ebenfalls Dateimanagement
42–46	2A–2E	Ebenfalls keine Ein- oder Ausgabe

Tabelle 16-1 Logische Einteilung der traditionellen DOS-Funktionsaufrufe

Warnung: **Einige DOS-Funktionen, insbesondere die Funktionen 1 bis 12, weisen zum Teil sehr merkwürdige Aspekte in der Konzeption auf. Das hat historische Gründe. Bei der Entwicklung der ersten DOS-Version wurde sehr viel Wert auf die Kompatibilität zu dem damals vorherrschenden Betriebssystem CP/M gelegt. Viele Einzelheiten von DOS und vor allem die Details der DOS-Funktionsaufrufe wurden auf entsprechende CP/M-Routinen abgestimmt. Das war damals eine wohldurchdachte Wahl, denn auf diese Weise war es einfach, die 8-bit-CP/M-Software für den 16-bit-PC und DOS zu konvertieren. Der erste Bruch mit der CP/M-Vergangenheit kam mit der DOS-Version 2.00, die zusätzliche Funktionen enthielt. Lesen Sie hierzu auch Kapitel 17.**

Funktion Dez	Hex	Erklärung
0	0	Programm beenden
1	1	Tastatureingabe mit Echo
2	2	Bildschirmausgabe
3	3	Serielle Eingabe
4	4	Serielle Ausgabe
5	5	Druckerausgabe
6	6	Direkte Tastatur/Bildschirm-Ein-/Ausgabe
7	7	Direkte Tastatureingabe ohne Echo
8	8	Tastatureingabe ohne Echo
9	9	Zeichenkette (String) darstellen
10	A	Gepufferte Tastatureingabe
11	B	Tastatureingabestatus prüfen
12	C	Tastaturpuffer löschen und Funktion ausführen
13	D	Laufwerks-Reset
14	E	Standardlaufwerk bestimmen
15	F	Datei öffnen
16	10	Datei schließen
17	11	Nach erster passender Datei suchen
18	12	Nach nächster passender Datei suchen
19	13	Datei löschen
20	14	Datensatz sequentiell lesen
21	15	Datensatz sequentiell schreiben
22	16	Datei anlegen
23	17	Datei umbenennen
24	18	Wird intern von DOS benutzt
25	19	Standardlaufwerk feststellen
26	1A	Diskettentransferbereich (DTA) festlegen
27	1B	FAT-Informationen des Standardlaufwerkes lesen
28	1C	FAT-Informationen eines beliebigen Laufwerkes lesen
33	21	Datensatz wahlfrei lesen
34	22	Datensatz wahlfrei schreiben
35	23	Dateilänge feststellen
36	24	Feld für wahlfreien Zugriff setzen
37	25	Interruptvektor setzen
38	26	Programmsegment anlegen
39	27	Datensätze wahlfrei lesen
40	28	Datensätze wahlfrei schreiben
41	29	Dateiname durchsuchen
42	2A	Datum ablesen
43	2B	Datum stellen
44	2C	Tageszeit ablesen
45	2D	Tageszeit stellen
46	2E	Diskettenschreibverifikation setzen

Tabelle 16-2 Die traditionellen DOS-Funktionsaufrufe, die durch Interrupt 33 (hex 21) aufgerufen und in Register AH gewählt werden

Auf den nächsten Seiten werden wir die 46 traditionellen DOS-Routinen, die in allen DOS-Versionen vorhanden sind, ausführlich besprechen.

16.1.1 Funktion 0: Programm beenden

Die DOS-Funktion 0 wird verwendet, um ein Programm zu beenden und die Kontrolle an DOS zurückzugeben. Sie ist funktionell identisch mit Interrupt 32 (hex 20), den wir in Kapitel 15.1.1 behandelt haben. Beide Routinen können gleichberechtigt verwendet werden, um ein Programm zu beenden.
Ab der DOS-Version 2.00 steht neben der Funktion 0 eine überarbeitete Routine zur Verfügung. Sie wird mit Interrupt 76 (hex 4C) aufgerufen und hinterläßt einen Fehlercode im AL-Register, wenn das Programm beendet wird. Batch-Dateien können über das DOS-Kommando ERRORLEVEL auf den Code zugreifen. Wenn also festgehalten werden soll, welche Fehler bei der Programmbeendigung auftraten, verwenden Sie statt der Funktion 0 die Funktion 76 (hex 4C). Weitere Erläuterungen zu Funktion 76 finden Sie in Kapitel 17.
Genau wie beim DOS-Interrupt 32 werden auch bei Funktion 0 keine Dateien geschlossen, bevor die Programmausführung abgebrochen wird. Es sollten daher die Funktionen 16 oder 62 (zum Schließen von Dateien) vor Verwendung von Funktion 0 bzw. Interrupt 32 aufgerufen werden. Dadurch wird sichergestellt, daß die korrekte Dateilänge in das Dateiverzeichnis eingetragen wird. Außerdem ist zu beachten, daß die PSP-Adresse in Register CS stehen muß, bevor das Programm beendet wird. Wie Sie vielleicht den Erläuterungen in Kapitel 15 entnommen haben, enthält das PSP die Adresse, bei der der Prozessor nach Programmende weiterarbeitet.

16.1.2 Funktion 1: Tastatureingabe mit Echo

Funktion 1 wartet, bis die Eingabe eines Zeichens auf dem Standardeingabegerät erfolgt und legt das Zeichen im AL-Register ab. Sie sollten diese Funktion mit den anderen Tastatureingabefunktionsaufrufen, besonders mit den Funktionen 6, 7 und 8, vergleichen.
So arbeitet Funktion 1: Ein Tastendruck, der ein ASCII-Zeichen erzeugt, wird als einzelnes Byte in AL geladen und gemeldet. Die 97 Tastenanschläge, deren Ergebnis kein ASCII-Zeichen ist (siehe Kapitel 6.2.2), generieren zwei Bytes, die durch zwei aufeinanderfolgende Aufrufe der Funktion gelesen werden können.
Üblicherweise wird Funktion 1 verwendet, um zu testen, ob in AL eine Null steht. Ist das nicht der Fall, liegt ein ASCII-Zeichen vor. Andernfalls handelt es sich um ein Sonderzeichen (das festgehalten werden sollte) und der Funktionsaufruf sollte unmittelbar im Anschluß wiederholt werden, um den Pseudoauswahlcode, der zu der gedrückten Sondertaste gehört, zu erhalten. In Kapitel 6.2 finden Sie eine Auflistung der Tastenanschläge und Codes. Wie bei allen DOS-Tastatureingaberoutinen ist der Auswahl-

code nicht direkt verfügbar. Man kann ihn über BIOS-Routinen im Hilfs-Byte erhalten (siehe Kapitel 6.2.2).
Die verschiedenen DOS-Tastaturfunktionen werden vorrangig nach drei Kriterien unterteilt: ob sie auf eine Eingabe warten oder melden, wenn keine Eingabe stattgefunden hat; ob sie die Eingabe auf dem Bildschirm darstellen und ob die Unterbrechung mit Break abgefragt wird. Bedenken Sie, daß nur einige der Standard-DOS-Operationen eine Unterbrechung mit Ctrl-Break oder Ctrl-C erkennen. Mit der Version 2.00 wurde aber der Befehl BREAK ON eingeführt, der DOS in die Lage versetzt, eine Unterbrechung unter allen Umständen und zu jedem Zeitpunkt durchzuführen. Die Funktion 1 erfüllt alle drei angeführten Punkte: Sie wartet auf eine Eingabe, gibt sie an den Bildschirm weiter (Echofunktion) und führt bei einem Break-Tastendruck den Unterbrechungsadreßinterrupt 36 aus.

Hinweis: Ab DOS-Version 2.00 ist die Tastaturfunktion 1 mit der Standardeingabenumgebung von DOS verbunden. Die Eingabeabfrage ist standardmäßig auf die Tastatur gerichtet, kann aber auf jedes andere Eingabegerät umgelenkt werden.

Soll das Warten auf eine Eingabe unterbunden werden, verwenden Sie Funktion 11, die ebenfalls meldet, ob eine Eingabe vorliegt oder nicht. Variationen der Funktion 1 finden Sie bei den Funktionsaufrufen 8 und 12.

16.1.3 Funktion 2: Bildschirmausgabe

Funktion 2 schreibt ein einzelnes ASCII-Zeichen auf den Bildschirm (oder ab DOS-Version 2.00 auf das festgelegte Standardausgabegerät). Das Zeichen muß im Register DL abgelegt werden.
Die Funktion bearbeitet die Mehrzahl der ASCII-Kontrollzeichen wie Backspace oder Wagenrücklauf korrekt. Im Falle des Backspace-Zeichens wird der Cursor rückwärts in Richtung des Zeilenbeginns gesetzt. Dabei wird nicht, wie manchmal behauptet, ein Leerzeichen (hex 20) ausgegeben, nachdem der Cursor bewegt wurde, denn das würde ja jedes vorhergehende Zeichen löschen. Tatsache ist, daß die übersprungenen Zeichen vollständig erhalten bleiben.

16.1.4 Funktion 3: Serielle Eingabe

Funktion 3 holt ein Zeichen vom Standardhilfsgerät (üblicherweise AUX: oder COM1:) in das Register AL. Mit Hilfe des DOS-Kommandos MODE kann die Eingabe auf jedes andere Gerät umgelenkt werden, z.B. auf COM2:. Die Eingabequelle ist für gewöhnlich der erste RS-232-Port.

Hinweis: Die Routine wartet auf eine Eingabe. Sie meldet nicht den Fehlerstatus, der bei der Eingabe über einen seriellen Port sehr wichtig sein kann. Um den Status zu ermitteln, müssen die entsprechende Routinen des ROM-BIOS eingesetzt werden. Mehr hierzu in Kapitel 12.1.

16.1.5 Funktion 4: Serielle Ausgabe

Funktion 4 gibt das Zeichen, das in Register DL steht, auf dem Standardhilfsgerät aus. Die Anmerkungen unter Funktion 3 gelten sinngemäß.

16.1.6 Funktion 5: Druckerausgabe

Funktion 5 gibt das Byte aus dem Register DL auf den Standarddrucker (PRN: oder LPT1:) aus. Mit dem DOS-Kommando MODE kann aber auch ein anderer Drucker bestimmt werden. Erfolgt keine Umleitung der Ausgabe, ist der Standarddrucker immer der erste Paralleladapter, auch wenn ein serieller Port für die Druckerausgabe verwendet wird.

16.1.7 Funktion 6: Direkte Tastatur/Bildschirm-Ein-/Ausgabe

Funktion 6 ist eine komplexe und etwas realitätsfremde Routine, die die Tastatureingabe und die Bildschirmausgabe verknüpft. Ab DOS 2.00 wird die Ein- und Ausgabe nicht mit Tastatur und Bildschirm verbunden, sondern mit den Standardein- und ausgabegeräten (die allerdings Tastatur und Bildschirm darstellen, wenn keine anderen Angaben gemacht werden).
Das AL-Register wird zur Eingabe verwendet, das DL-Register zur Ausgabe. AL steht zur Aufnahme eines Eingabezeichens bereit, wenn der Wert in DL gleich 255 (hex FF) ist. Die Nullflagge (ZF) zeigt an, ob eine Eingabe ansteht. Für ZF gleich 1 gilt: Es liegt keine Eingabe vor; für ZF gleich 0: Ein Eingabe-Byte ist in AL vorhanden. Hat DL einen anderen Wert als 255, wird angenommen, daß das Register ein Ausgabezeichen enthält, das auf Anforderung über DL auf das Standardausgabegerät gegeben wird.
Die Routine wartet nicht auf eine Tastatureingabe, sie bringt das eingegebene Zeichen nicht auf den Bildschirm und die Break-Tastaturabfrage ist nicht aktiv (siehe Funktion 1).
Vergleichen Sie diese Routine mit den Funktionen 1, 7 und 8. Funktion 12 ist eine Variation von Routine 6.

16.1.8 Funktion 7: Direkte Tastatureingabe ohne Echo

Die Funktion 7 wartet auf eine Eingabe über das Standardeingabegerät und übergibt das Zeichen, sobald verfügbar, an das Register AL. Das Zeichen wird nicht auf dem Bildschirm dargestellt, der Break-Tastenanschlag bleibt unbeachtet.
Funktion 7 arbeitet genau wie Funktion 1: ASCII-Zeichen werden als einzelne Bytes nach AL gebracht und es erfolgt eine sofortige Meldung. Die 97 Sondertastenanschläge, die keine ASCII-Zeichen erzeugen (siehe Kapitel 6.2.2), werden dekodiert, indem die Funktion zweimal unmittelbar hintereinander aufgerufen wird.
Auch hier wird für gewöhnlich abgetestet, ob AL gleich 0 ist. Falls AL ungleich 0 ist, liegt ein normales ASCII-Zeichen, ansonsten (AL gleich 0) ein Sonderzeichen vor. Bei einem Sonderzeichen sollte das Zeichen gespeichert und der Funktionsaufruf wiederholt werden, damit der Pseudoauswahlcode ermittelt werden kann. In Kapitel 6.2 finden Sie eine Auflistung der Tastendrücke und der Codes mit Beschreibung. Wie bei allen DOS-Tastatureingaberoutinen ist der Auswahlcode eines Zeichens nicht direkt verfügbar, kann aber mit den entsprechenden BIOS-Routinen ermittelt werden (siehe Kapitel 6.2.2.).
Vergleichen Sie die Routine mit den Funktionen 1, 6 und 8. Wenn Sie unnötige Wartezeiten auf Tastatureingaben vermeiden wollen, verwenden Sie Funktion 11. Diese meldet, ob ein Zeichen anliegt oder nicht. Funktion 12 stellt eine abgewandelte Form der Routine 7 dar.

16.1.9 Funktion 8: Tastatureingabe ohne Echo

Funktion 8 wartet auf eine Eingabe, erzeugt kein Zeichen auf dem Bildschirm und bearbeitet einen Break-Tastendruck. Sie arbeitet entsprechend der Funktion 1 mit dem Unterschied, daß der Bildschirm (oder das Standardausgabegerät) nicht angesprochen wird.
Die Details entnehmen Sie bitte den Erläuterungen zu Funktion 1. Vergleichen Sie Funktion 8 mit den Funktionen 1, 6 und 7. Die Wartezeit auf Tastendrücke können Sie vermeiden, indem Sie vor Funktion 8 erst Funktion 11 aufrufen. Diese meldet, ob ein Zeichen anliegt oder nicht. Eine Variante von Funktion 8 findet sich in Funktion 12 wieder.

16.1.10 Funktion 9: Zeichenkette (String) darstellen

Funktion 9 sendet eine Zeichenkette, einen String, an den Bildschirm oder das Standardausgabegerät. Das Registerpaar DS:DX enthält die Adresse des Strings. Ein "$"-Zeichen, CHR$(36), kennzeichnet das Stringende.

Obwohl die Routine grundsätzlich sehr viel einfacher einzusetzen ist als die byteweise arbeitenden Anzeigeroutinen 2 und 6, wird sie dennoch nicht häufig benutzt. Sie hat nämlich einen großen Nachteil: Die Zeichenkette muß durch das abbildbare Zeichen "$" abgeschlossen werden. Sie können mit der Routine also kein "$"-Zeichen ausgeben. Das ist, wie so vieles im DOS-Bereich, ein Überbleibsel der CP/M-Kompatibilität. In Programmen, bei denen Sie die Verwendung von Dollarzeichen nicht von vornherein sicher ausschließen können, sollten Sie die Funktion vermeiden. Bei einem deutschen Programm mag das nicht unbedingt zu einem Problem werden, bedenken Sie aber, welcher Aufwand sich ergeben kann, wenn ein Programm auf amerikanische Verhältnisse übertragen werden soll.
Die erweiterten DOS-Funktionen (siehe Kapitel 17) verwenden CHR$(0) als Endmarke, das entspricht dem Standard des Betriebssystems UNIX und der Programmiersprache C.

16.1.11 Funktion 10 (hex A): Gepufferte Tastatureingabe

Funktion 10 ist ein wirkungsvolles "Werkzeug" zur Integration der Editiertasten in Programme. Die Routine liest einen kompletten String ein und übergibt ihn dem Programm als Ganzes, also nicht Byte für Byte. Unter der Annahme, daß die Eingabe direkt von der Tastatur kommt und nicht von irgendwoher umgeleitet wird, stehen dem Computerbenutzer alle von DOS unterstützten Editiertasten uneingeschränkt zur Verfügung. Wird die Return-Taste gedrückt (oder auf andere Weise ein CHR$(13) erzeugt), ist die Eingabe beendet und der gesamte String wird dem Programm übergeben.
Die Anwendung dieser Routine ist recht bequem, da man keine eigenen Routinen schreiben muß, die die Tastatureingabe überwachen. Für den Anwender ergibt sich der Vorteil, die bekannten DOS-Editierfunktionen zur Verfügung zu haben.
Die Routine benötigt einen Eingabepuffer, in dem die Zeichenkette abgelegt wird. Das Registerpaar DS:DX zeigt auf diesen Puffer. Das erste Byte des Puffers enthält seine Länge in Bytes. Im zweiten Byte steht die jeweils aktuelle Anzahl eingegebener Bytes. Der Eingabestring, der nur aus ASCII-Zeichen bestehen darf, wird ab der dritten Speicherstelle gespeichert. Das Ende der Zeichenkette wird durch einen Wagenrücklauf CHR$(13) gekennzeichnet. Der Wagenrücklauf muß im Puffer stehen und belegt ein Byte. Das zweite Byte des Puffers zählt den Wagenrücklauf aber nicht mit.
Aus den Regeln ergibt sich, daß der Eingabepuffer aus maximal 255 Bytes und der längste String, den DOS übermitteln kann, aus 254 Bytes besteht. Da die ersten zwei Bytes des Puffers für Statusinformationen benötigt werden, ist der tatsächlich nutzbare Arbeitsbereich des Puffers

nochmals um zwei Bytes geringer. Das erklärt vielleicht einige Besonderheiten bei der Eingabe sowohl bei DOS als auch in BASIC.
Erfolgt eine Eingabe, die die Länge des Puffers übersteigt, werden solange keine weiteren Zeichen akzeptiert (das Gerät piept bei jedem zusätzlichen Tastendruck), bis ein Wagenrücklauf erfolgt und der String abgeschlossen ist.
Sie können die Tastenpufferung testen, indem Sie bei der Eingabe mitzählen, wie viele Zeichen der DOS-Kommandointerpreter akzeptiert. Das wird Ihnen bestätigen, daß der Interpreter mit einem Puffer von 130 Bytes Gesamtlänge, das heißt, mit einem Arbeitsbereich von 128 Bytes arbeitet. DOS nimmt ab dem 127. Byte keine weitere Tastendrücke außer dem Wagenrücklauf (Return-Taste) mehr an.
Funktion 12 stellt eine Variante der Funktion 10 zur Verfügung.

16.1.12 Funktion 11 (hex B): Tastatureingabestatus prüfen

Mit Funktion 11 kann festgestellt werden, ob eine Eingabe von der Tastatur (oder vom Standardeingabegerät) anliegt. Die Routine ist besonders im Zusammenhang mit *den* Funktionen nützlich, die auf eine Eingabe warten. Das sind die Funktionen 1, 7 und 8. Eine anstehende Eingabe wird durch den Wert 255 (hex FF) in AL gemeldet. Liegt keine Eingabe vor, ist AL gleich 0.
Die normale Break-Tastaturabfrage ist aktiv. Eine Erklärung hierzu finden Sie bei Funktion 1.

16.1.13 Funktion 12 (hex C): Tastaturpuffer löschen und Funktion ausführen

Funktion 12 löscht den Tastatureingabepuffer und ruft dann eine der folgenden fünf DOS-Routinen auf: Funktion 1, 6, 7, 8 oder A. Der Wert des AL-Registers bestimmt, welche dieser Funktionen gewählt wird. Da nach dem Aufruf von Funktion 12 der Eingabepuffer leer ist, muß die Anschlußroutine (1, 6, 7, 8 oder A) warten, bis erneut eine Taste gedrückt wird. Das ist z.B. für Sicherheitsabfragen nützlich. Bei "Wollen Sie Ihre Daten löschen? (J=Ja)" sollte vermieden werden, daß ein vorangegangener Druck auf die Taste J im Zeitpunkt der Abfrage wirksam wird.

Beachten Sie: Da Funktion 6 unterstützt wird, muß der nachfolgende Funktionsaufruf nicht notwendigerweise eine Tastatureingabe sein, eine Bildschirmausgabe ist beispielsweise auch möglich.

16.1.14 Funktion 13 (hex D): Laufwerks-Reset

Funktion 13 führt einen Laufwerks-Reset durch und löscht alle Dateipuffer. Es empfiehlt sich, zuvor die Funktionen 16 oder 62 aufzurufen, um die Dateien zu schließen. Andernfalls kann es passieren, daß eine falsche Dateilänge im Dateiverzeichnis abgespeichert wird.
Manchmal ist die Behauptung zu hören, die Routine lege Laufwerk A als Standardlaufwerk fest. Das ist falsch, die Laufwerkszuordnung bleibt unverändert.

16.1.15 Funktion 14 (hex E): Standardlaufwerk bestimmen

Funktion 14 legt fest, welches Laufwerk als Standardlaufwerk angesehen werden soll und meldet die Anzahl der angeschlossenen Laufwerke. Das Standardlaufwerk wird durch Register DL ausgewählt, wobei der Wert 0 dem Laufwerk A enspricht, der Wert 1 dem Laufwerk B usw. Die Anzahl der angeschlossenen Laufwerke wird in das Register AL gemeldet. Wenn DOS die Anzahl der installierten Laufwerke kennt, kann jede Nummer von 0 bis zu eins weniger als die Gesamtzahl das Standardlaufwerk angewählt werden.
Es gibt einige Dinge, die Sie wissen sollten, wenn sie diese Funktion verwenden wollen. Erstens ist die Laufwerksnumerierung von DOS lükkenlos. Zweitens: Ist nur ein einziges Laufwerk angeschlossen, simuliert DOS ein zweites mit der Nummer 1 (Laufwerk B). Und drittens können Sie den Buchstaben, der jedem Laufwerk zugeordnet ist, finden, indem Sie die Laufwerksnummer zum Wert des Buchstabens "A" CHR$(65) addieren. Für den außergewöhnlichen Fall, daß Sie mehr als 26 Laufwerke angeschlossen haben, mag das natürlich zu einigen sehr seltsamen "Buchstaben" führen.
Die Funktion 25 (hex 19) meldet die Nummer des Standardlaufwerkes. Die Funktionen 14 und 15 können problemlos verwendet werden, um die Anzahl der angeschlossenen Laufwerke festzustellen, das aktuelle Laufwerk wird dabei nicht verändert. Hier ein kleines Assemblerprogramm, das diese Aufgabe erfüllt:

```
MOV   AH,25    ;Standardlaufwerk feststellen
INT   33       ;Funktionsaufruf
MOV   DL,AL    ;Laufwerksnummer kopieren
MOV   AH,14    ;Standardlaufwerk bestimmen
INT   33       ;Funktionsaufruf
```

Nach der Abarbeitung des Assemblerprogramms enthält AL die Anzahl der Laufwerke, ohne daß das Standardlaufwerk verändert wurde. Um die

Nummer in den höchsten Laufwerksbuchstaben umzuwandeln, können wir folgenden Befehl hinzufügen:

```
ADD   AL,'A'-1
```

16.1.16 Funktion 15 (hex F): Datei öffnen

Funktion 15 öffnet eine Datei über den *Dateikontrollblock* (*File Control Block*, FCB), der von DOS generell für Dateifunktionen verwendet wird. Der FCB ist eine Datei mit 128 logischen Einträgen. Er enthält Informationen, die DOS benötigt, um auf die zum jeweiligen FCB zugehörige Datei zugreifen zu können. Das Registerpaar DS:DX verweist auf den FCB. In Kapitel 16.2 finden Sie nähere Erläuterungen zum Dateikontrollblock. DOS versucht aufgrund der Spezifikationen des FCB eine Datei zu öffnen. Das Ergebnis des Versuches wird in das Register AL gemeldet: 0 bedeutet Erfolg, 255 (hex FF) Mißerfolg.
Verwechseln Sie das Öffnen einer Datei (Funktion 15) nicht mit dem Anlegen einer Datei (Funktion 22). Eine Datei, die geöffnet werden soll, muß bereits existieren, das heißt, sie muß bereits angelegt worden sein. Üblicherweise wird man Funktion 15 verwenden, wenn eine Eingabedatei geöffnet werden soll. Für eine Ausgabedatei bietet sich eher Funktion 22 an.
Ist eine Datei geöffnet, werden mehrere Felder des FCB von DOS mit Informationen belegt. Wenn das Standardlaufwerk durch Funktion 14 festgelegt wurde, trägt DOS die Laufwerksnummer ein. Die Numerierung ist ungewöhnlich, Laufwerk A erhält die Nummer 1, nicht 0, wie bei den meisten Laufwerksoperationen. DOS füllt ebenfalls die Felder für Datum und Zeit auf und setzt den momentanen Block auf Null.
Beim Öffnen einer Datei setzt DOS die Datensatzlänge mit 128 (hex 80) fest. Im allgemeinen wird man diese Angabe nach eigenen Wünschen verändern, nachdem die Datei geöffnet ist. Viele Texteditierprogramme belassen es aber bei 128 Bytes, um die Ein und Ausgabe effizient abwikkeln zu können. Welche Ironie, EDLIN selbst, der DOS-Editor, verwendet diese Vorgabe nicht. Mehr über das Feld finden Sie in Kapitel 16.2.

16.1.17 Funktion 16 (hex 10): Datei schließen

Funktion 16 schließt eine Datei, wenn das Registerpaar DS:DX als Zeiger auf den FCB deutet. Ist die Operation geglückt, wird in das Register AL der Wert 0 gemeldet, andernfalls der Wert 255 (hex FF).
Nach dem Schreiben in eine Datei muß sie geschlossen werden, damit das Dateiverzeichnis auf den neuesten Stand korrigiert werden kann. DOS macht einen "intelligenten" Versuch, herauszufinden, ob die Datei, die

geöffnet wurde, auch die ist, die geschlossen werden soll. Zu diesem Zweck werden die Laufwerksspezifikationen im Dateiverzeichnis mit denen des geöffneten FCB verglichen. Das bietet einen gewissen Schutz gegen das Vermischen von Disketteninformationen. Dazu kann es kommen, wenn nach dem Öffnen einer Datei die Diskette gewechselt wird, ohne daß die Datei zuvor geschlossen wurde.

16.1.18 Funktion 17 (hex 11): Nach erster passender Datei suchen

Funktion 17 startet einen Suchvorgang nach einer Datei, die bestimmte Spezifikationen erfüllt. Das Registerpaar DS:DX zeigt auf den FCB, der den Dateinamen enthält, nach dem gesucht werden soll. Die Routine ermöglicht es, mit Dateinamen umzugehen, die die Dateigruppenzeichen "?" und "*" enthalten. Funktion 17 sucht nach der ersten Datei, die der Namensvorgabe entspricht, Funktion 18 sucht nach weiteren Dateien, die in das Muster passen.
Auch hier signalisiert AL den Erfolg der Operation mit 0 und das Mißlingen mit 255 (hex FF). Wenn eine passende Datei gefunden wird (AL gleich 0), bringt DOS den Namen der Datei in das entsprechende Feld des FCB, um die Datei auf das Öffnen vorzubereiten. Der FCB enthält außerdem Informationen, die die Funktion 18 zum Suchen der nächsten Dateien benötigt.

Achtung: Die Informationen, die die Funktion 18 braucht, um weitere Dateien zu finden, werden von jeder Dateioperation, die den FCB anspricht, zerstört. Wenn Sie mehrere Dateien suchen, aber auch Ein- und Ausgaben in den Dateien vornehmen wollen, müssen Sie die Suchinformationen, die im FCB enthalten sind, sichern.

Wenn der FCB eine FCB-Erweiterung hat (siehe Kapitel 16.2), können Sie Attribute spezifizieren, die in die Suche mit einbezogen werden. Die Attribut-Suche mit einer Kombination aus versteckten, System- oder Verzeichnisattribut-Bits bringt als Ergebnis normale Dateien und solche mit den gewünschten Attributen. Wird ein Datenträgerattribut angegeben, werden nur die Verzeichniseinträge mit dieser Spezifikation herausgesucht. Bei DOS-Versionen vor Version 2.00 können weder das Verzeichnis- noch das Datenträgerattribut genutzt werden. Die Attribute *Archiv* und *Nur-lesen* lassen sich in keiner DOS-Version als Suchkriterium verwenden.
Die Routine 17 ist auch dann sinnvoll einsetzbar, wenn nur eine einzige Datei gesucht wird. Sie ermöglicht es, daß in Programmen die Namenszeichen "*" und "?" verwendet werden dürfen, was oftmals sehr nützlich ist.
Hier eine Struktur, in der die Funktionen 17 und 18 benutzt werden, um mehrere Dateien zu finden und zu bearbeiten:

```
10 'Funktion 17 aufrufen: ersten passenden Dateinamen suchen
20 WHILE DATEI_GEFUNDEN
30 'FCB-Feld für nächste Suchoperation sichern
40 'Gefundene Datei bearbeiten
50 'FCB-Feld für nächste Suchoperation wieder herstellen
60 'Funktion 18 aufrufen: nächsten passenden Dateinamen suchen
70 WEND
```

16.1.19 Funktion 18 (hex 12): Nach nächster passender Datei suchen

Funktion 18 findet die nächste Datei im Dateiverzeichnis, die den in Funktion 17 festgelegten Kriterien entspricht. Lesen Sie bitte die Einzelheiten in der Beschreibung von Funktion 17 nach.

16.1.20 Funktion 19 (hex 13): Datei löschen

Funktion 19 löscht die Dateien, die dem FCB entsprechen, auf den das Registerpaar DS:DX zeigt. AL ist 0, wenn die Operation erfolgreich verlaufen ist und alle zutreffenden Verzeichniseinträge gelöscht wurden. AL gleich 255 (hex FF) signalisiert, daß keine passenden Verzeichniseinträge gefunden wurden.
Erläuterungen zum Dateikontrollblock FCB finden Sie in Kapitel 16.2.

16.1.21 Funktion 20 (hex 14): Datensatz sequentiell lesen

Funktion 20 liest den nächsten Datensatz einer Datei, wobei sich "nächsten" auf die sequentielle Reihenfolge der Datensätze bezieht. Bevor die Funktion aufgerufen wird, muß der Zeiger DS:DX auf den FCB der Datei verweisen. Die sequentielle Datensatz- oder Eintragsnummer wird den Werten des momentanen Blocks und des momentanen Datensatzfeldes des FCB entnommen. Dann wird der Datensatz gelesen. Bei einem erfolgreichen oder zumindest teilweise erfolgreichen Versuch werden die Daten in den derzeitigen Diskettentransferbereich übertragen (siehe auch unter Funktion 26).
DOS erhöht die Datensatzadreßfelder des Dateikontrollblockes nach jedem Lesevorgang automatisch, um das sequentielle Lesen einer Datei zu beschleunigen. Man kann die Adreßfelder auch selbst ändern, was bei sequentiellen Dateien allerdings nicht empfehlenswert ist. Um wahlfrei auf eine Datei zuzugreifen, sollten die Funktionen 33 und 34 verwendet werden. In Kapitel 16.2 finden Sie eine Erläuterung der Unterschiede der Datensatznummern bei sequentiellem und wahlfreiem Zugriff.
Der Erfolg der Funktion ist aus Register AL ersichtlich. Dort befindet sich der Wert 0, wenn der Lesevorgang komplett erfolgreich war. AL

gleich 1 signalisiert das Ende einer Datei, es wurden keine Daten gelesen. AL gleich 2 bedeutet, daß Daten übertragen worden wären, wenn der Diskettentransferbereich groß genug gewesen wäre, das heißt, für eine Übertragung der gesamten Daten war nicht genügend Platz vorhanden. Der Wert 3 in Register AL gibt an, daß ein Teil der Daten gelesen wurde, bis das Dateiende erreicht wurde. In diesem Fall wird der Eintrag im Speicher mit Nullen aufgefüllt.

16.1.22 Funktion 21 (hex 15): Datensatz sequentiell schreiben

Funktion 21 schreibt sequentiell einen Datensatz in eine Datei und ist das Gegenstück zur eben besprochenen Funktion 20. Das Registerpaar DS:DX zeigt auf den FCB, der die Nummer des Datensatzes entsprechend der sequentiellen Numerierung enthält. DOS schreibt die Daten, die im DTA stehen (siehe Funktion 26), in den entsprechenden Datensatz auf dem Datenträger.
In AL ist der Status der Operation abfragbar. Wurde der Schreibvorgang erfolgreich durchgeführt, steht dort eine 0. Eine 1 bedeutet, daß die Diskette voll ist und keine Daten mehr dazugeschrieben werden können. Der Wert 2 zeigt an, daß im Diskettentransfersegment nicht genug Platz vorhanden ist, um den Datensatz zu schreiben. Das DOS-interne Diskettentransfersegment muß groß genug sein, um die Einträge aus dem Diskettentransferbereich aufnehmen zu können.
Die Routine schreibt die Datensätze in logischer, nicht in physikalischer Reihenfolge. DOS puffert die Ausgabedaten, bis genügend Daten für einen kompletten Diskettensektor bereitstehen; erst dann werden die Daten auf den Datenträger geschrieben. Wenn eine Datei geschlossen wird, können eventuell im Puffer noch Daten liegen, die ein Programm zum Schreiben vorgesehen hat, die aber noch nicht übertragen wurden. Das kann zum Problem werden, wenn ein Programm vorzeitig abgebrochen wird.

16.1.23 Funktion 22 (hex 16): Datei anlegen

Funktion 22 sucht nach einem Verzeichniseintrag oder legt einen neuen Eintrag im Verzeichnis an. Die Routine sucht den ersten Eintrag, der den vorgegebenen Spezifikationen entspricht. Wird keiner gefunden, nimmt sie den ersten leeren Eintrag und öffnet die Datei. In DS:DX steht wie üblich der Zeiger auf den FCB. Funktion 22 wird überwiegend zum Öffnen von Ausgabedateien benutzt, während für Eingabedateien zumeist Funktion 15 verwendet wird.
Der Status ist in AL ablesbar. Eine erfolgreiche Ausführung wird durch AL gleich 0 angezeigt. AL gleich 255 (hex FF) signalisiert einen Fehler,

der normalerweise dadurch entsteht, daß das Verzeichnis voll ist und daher kein leerer Eintrag gefunden wurde. Wenn eine Datei mit dieser Funktion geöffnet wird, wird die Dateilänge auf Null gesetzt. Wir können neue Daten hinzufügen und anschließend die Funktion 16 zum Schließen der Datei ausführen lassen. In dem Fall sind die alten Inhalte der Datei verloren. Das passiert nicht, wenn wir zum Öffnen die Funktion 15 verwenden, die das Dateilängenfeld des FCB auf den Wert setzt, der im Verzeichniseintrag gefunden wurde.

16.1.24 Funktion 23 (hex 17): Datei umbenennen

Funktion 23 ändert den Namen einer Datei in einem modifizierten FCB, auf den das Registerpaar DS:DX zeigt. Der FCB wird zur Namensänderung anders als sonst behandelt. Obwohl Laufwerk und ursprünglicher Dateiname an den normalen Positionen stehen, beginnt der neue Dateiname bei Offset 16 des FCB, wo eigentlich das Dateilängenfeld beginnt. Lesen Sie hierzu auch in Kapitel 16.2 nach.
Wenn der neue Dateiname Dateigruppenzeichen wie "*" oder "?" enthält, werden die alten Zeichen an diesen Stellen in den neuen Namen übernommen.
Eine erfolgreiche Durchführung wird durch AL gleich 0 angezeigt, AL gleich 255 (hex FF) bedeutet, daß entweder keine Dateinamen gefunden wurden, die umbenannt werden konnten oder daß der neue Dateiname bereits verwendet wird.

16.1.25 Funktion 24 (hex 18): Wird intern von DOS benutzt

Funktion 24 und die Funktionen 29 bis 32 werden von DOS für interne Zwecke verwendet, Sie sollten die Funktionsaufrufe nicht in Programme einbauen. Zwar gibt es Informationen über diese Funktionen, bedenken Sie aber, daß zukünftige DOS-Versionen die Routinen vielleicht gar nicht mehr enthalten. Es empfiehlt sich, grundsätzlich nur DOS-Funktionen zu verwenden, die "offiziell" für Programmierer freigegeben sind. Das kann Ihnen in der Zukunft viel Ärger ersparen.

16.1.26 Funktion 25 (hex 19): Standardlaufwerk feststellen

Funktion 25 meldet die Nummer des Standardlaufwerkes in Register AL. Laufwerk A hat die Nummer 0, Laufwerk B die Nummer 1 usw. Ein Beispiel für die Verwendung dieser Funktion in einem Assemblerprogramm finden Sie unter Funktion 14.

16.1.27 Funktion 26 (hex 1A): Diskettentransferbereich (DTA) festlegen

Funktion 26 bestimmt die Position des Diskettentransferbereichs (DTA), der von DOS für die Datei-Ein-/Ausgabe verwendet wird. Das Registerpaar DS:DX zeigt auf den Bereich. Im PSP steht ab Offset hex 80 ein Standard-DTA von 128 Bytes Länge bereit.

16.1.28 Funktion 27 (hex 1B): FAT-Informationen des Standardlaufwerkes lesen

Funktion 27 gibt uns Schlüsselinformationen über die Diskette, die sich im Standardlaufwerk befindet. Die Funktion 28 erfüllt dieselbe Aufgabe für ein beliebiges Laufwerk. Die Funktion 54, die im nächsten Kapitel behandelt wird, stellt eine nahezu identische Routine dar.
Nach dem Aufruf von Funktion 27 stehen folgende Informationen zur Verfügung: AL enthält die Anzahl der Sektoren pro Belegungseinheit (1 Sektor für einseitige Diskette, 2 für zweiseitige Disketten), CX enthält die Länge der Sektoren in Byte (für alle normalen IBM-PC-Formate sind das 512 Bytes pro Sektor), DX enthält die Gesamtzahl der Belegungseinheiten (Cluster) auf der Diskette und das Registerpaar DS:BX zeigt auf ein Byte, aus dem das Diskettenformat abgelesen werden kann. Einzelheiten über die FAT finden Sie in Kapitel 5.5.4. Bei den DOS-Versionen vor 2.00 zeigt DS:BX zugleich auf den Anfang der FAT, da das Format-Byte das erste Feld der FAT darstellt. Ab Version 2.00 wird die FAT nicht in jedem Fall zusammenhängend abgelegt, so daß man nicht davon ausgehen kann, daß DS:BX auf den FAT-Anfang zeigt, sondern nur auf ein einzelnes Byte (Format-Byte).

Beachten Sie: Die Funktion setzt DS:BX auf das FAT-Format-Byte, das sich außerhalb des Datensegmentbereiches befindet. Das Datensegmentregister DS wird also für einen anderen als den üblichen Zweck eingesetzt. Viele Programme nehmen darauf aber keine Rücksicht und verursachen Schwierigkeiten. Um das Problem zu umgehen, sollten Sie den Inhalt des DS-Registers abspeichern, bevor Funktion 27 aufgerufen wird, und danach wieder den ursprünglichen Wert von DS herstellen.

Ein Beispiel dazu:

```
PUSH    DS          ;DS-Adresse sichern
MOV     AH,27       ;Funktion 27 anwählen
INT     33          ;DOS-Funktionsaufruf
MOV     AH,[BX]     ;FAT-Format-Byte holen
POP     DS          ;DS-Adresse zurückspeichern
```

Das DS-Problem zeigt, wie schnell ein unüberlegter Programmentwurf zu Schwierigkeiten führen kann. Würde die Funktion das Extrasegmentregister ES statt des Registers DS benutzen, bräuchte man sich keine Gedanken über die Registerinhalte zu machen.

16.1.29 Funktion 28 (hex 1C): FAT-Informationen eines beliebigen Laufwerkes lesen

Funktion 28 arbeitet genau wie Funktion 27, bezieht sich aber nicht nur auf das Standardlaufwerk, sondern auf ein beliebiges angeschlossenes Laufwerk. Welches Laufwerk angesprochen werden soll, wird in DL spezifiziert. Der Wert 0 entspricht dem Standardlaufwerk, 1 dem Laufwerk A, 2 dem Laufwerk B usw. Beachten Sie, daß hier also nicht die normale Numerierung, Laufwerk A gleich 0, Laufwerk B gleich 1 usw. verwendet wird.

16.1.30 Funktion 33 (hex 21): Datensatz wahlfrei lesen

Funktion 33 liest wahlfrei einen Datensatz einer Datei. Der Zeiger DS:DX wird auf den Dateikontrollblock FCB gesetzt und die Datensatznummer in dem FCB-Feld für wahlfreien Zugriff abgelegt. Durch den Aufruf der Routine werden die Daten des Datensatzes in den aktuellen DTA gelesen. AL gibt den Status mit dem gleichen Code wider, der auch bei sequentiellem Lesen gilt. AL gleich 0 bedeutet, daß die Leseoperation erfolgreich durchgeführt wurde, AL gleich 1 zeigt an, daß das Dateiende erreicht und keine Daten gelesen wurden. Nicht ausreichender Platz im Diskettentransfersegment wird durch den Wert 2 signalisiert und 3 bedeutet, daß das Ende einer Datei erreicht wurde und ein Teil der Daten gelesen werden konnte.

Hinweis: Programme, die die Leseroutine für wahlfreien Zugriff verwenden, müssen kontinuierlich das Feld für wahlfreien Zugriff im FCB auf die entsprechenden Datensätze setzen. Bei der Routine für sequentielles Lesen richtet DOS die FCB-Felder für sequentiellen Zugriff nach jeder Leseoperation automatisch auf den folgenden Datensatz. Oftmals werden beide Funktionen miteinander verknüpft, indem zuerst ein wahlfreier Zugriff erfolgt und danach sequentiell weitergelesen wird.

Vergleichen Sie Funktion 33 mit Funktion 39, die mehrere Datensätze wahlfrei lesen kann, und mit Funktion 20, die Daten sequentiell liest. Unter Funktion 36 finden Sie Näheres über das FCB-Feld für wahlfreien Zugriff.

16.1.31. Funktion 34 (hex 22): Datensatz wahlfrei schreiben

Funktion 34 schreibt in einen beliebigen Datensatz einer Datei. Das Registerpaar DS:DX muß auf den Dateikontrollblock FCB der Datei zeigen, das FCB-Feld für wahlfreien Zugriff auf den Datensatz, in den geschrieben werden soll. Anschließend werden die Daten des aktuellen DTA auf Diskette übertragen.
AL enthält den gleichen Statuscode wie beim sequentiellen Schreiben von Datensätzen. Der Wert 1 bedeutet, daß die Diskette voll ist, der Wert 2 zeigt an, daß zu wenig Platz im Diskettentransfersegment vorhanden ist und der Wert 0 signalisiert, daß die Operation erfolgreich durchgeführt wurde.

Hinweis: Auch hierbei muß das Programm das FCB-Feld für wahlfreien Zugriff ständig neu setzen, während die Felder für sequentiellen Zugriff von DOS automatisch auf den nächstfolgenden Datensatz gerichtet werden. Aus diesem Grund werden wahlfreies und sequentielles Schreiben oft miteinander kombiniert. Zuerst erfolgt ein wahlfreier Zugriff, dann werden die Datensätze sequentiell geschrieben.

Vergleichen Sie Funktion 34 mit Funktion 40, die mehrere Datensätze wahlfrei schreiben kann, und auch mit Funktion 21, die für das sequentielle Schreiben von Datensätzen benutzt wird. Mehr über das FCB-Feld für wahlfreien Zugriff finden Sie unter Funktion 36.

16.1.32 Funktion 35 (hex 23): Dateilänge feststellen

Funktion 35 meldet die Länge einer Datei, das heißt, die Anzahl der in der Datei enthaltenen Datensätze. DS:DX muß auf den FCB der Datei zeigen. Der FCB sollte aber vor dem Funktionsaufruf nicht geöffnet werden. Die Datensatzlänge wird im entsprechenden FCB-Feld spezifiziert. Wird die Datensatzlänge auf ein Byte festgelegt, erscheint als Resultat die Länge der Datei in byte.
Bei erfolgreicher Durchführung der Operation wird AL auf 0 gesetzt und die Länge der Datei in den FCB eingetragen. Wird die Datei nicht gefunden, steht in AL der Wert 255 (hex FF).

16.1.33 Funktion 36 (hex 24): Feld für wahlfreien Zugriff setzen

Funktion 36 setzt das Feld für wahlfreien Zugriff, so daß es mit dem momentanen sequentiellen Block und den Datensatzfeldern des FCB übereinstimmt. Die Funktion erleichtert das Umschalten von sequentiellem auf wahlfreien Zugriff beträchtlich. Das DS:DX-Registerpaar zeigt auf den FCB einer offenen Datei.

16.1.34 Funktion 37 (hex 25): Interruptvektor setzen

Mit Funktion 37 kann ein Interruptvektor gesetzt werden. Das Registerpaar DS:DX enthält die Vektoradresse der Interruptroutine und AL die Nummer des Interrupts. Der Interruptvektor belegt vier Bytes in der Vektortabelle und wird über die Interruptnummer mit dem Befehl INT aufgerufen.
In einem Programm kann natürlich auch ohne Funktion 37 ein Interruptvektor festgelegt werden. Der Funktionsaufruf befreit jedoch von einer Reihe von Vorbereitungen und Schutzmaßnahmen, die üblicherweise mit der Veränderung von Interruptvektoren verbunden sind. Bei Vektoren, die statt auf Tabellen auf Interruptroutinen zeigen, wäre der Segmentteil normalerweise das aktuelle Codesegment (CS), das nach DS übertragen werden muß. Beachten Sie auch hier die unsinnige Wahl des DS-Registers, wo das ES-Register angebrachter wäre.
Wie der Inhalt eines Interruptvektors eingesehen werden kann, steht im nächsten Kapitel bei Funktion 53 (hex 35).

16.1.35 Funktion 38 (hex 26): Programmsegment anlegen

Funktion 38 wird verwendet, um ein neues Programmsegment anzulegen als Vorbereitung für ein separat zu ladendes Unterprogramm oder Overlay-Programm, das ausgeführt werden soll. In DX wird die Segmentadresse des neuen Programms abgelegt. Das Programmsegmentpräfix (PSP) des aktuellen Programms wird in die ersten 256 (hex 100) Bytes des neuen Segmentbereiches kopiert und stellt dort ein neues PSP dar. Das neue PSP wird mit neuem Speicher und Interruptvektorinformation auf den aktuellen Stand gebracht. Danach können die normalen DOS-Routinen verwendet werden, um ein COM-Programm in den Bereich unmittelbar hinter das PSP zu laden. Ab der DOS-Version 2.00 ist der ganze Komplex *Overlay-Programmierung* allerdings sehr viel einfacher.
Lesen Sie im nächsten Kapitel nach, insbesondere bei Funktion 75. Eine Erklärung des Programmsegmentpräfixes (PSP) finden Sie in Kapitel 15.3.

16.1.36 Funktion 39 (hex 27): Datensätze wahlfrei lesen

Im Unterschied zu Funktion 33 kann die Funktion 39 nicht nur einen, sondern mehrere Datensätze einlesen. Der Zugriff auf den ersten Datensatz erfolgt wahlfrei, die anderen Datensätze müssen sequentiell folgen. DS:DX zeigt auf den FCB, dem die Datensatznummer, ab der gelesen werden soll, entnommen wird. Das Register CX enthält die Anzahl der zu lesenden Datensätze, der Wert sollte über Null liegen.

Der Statuscode entspricht dem von Funktion 33. AL gleich 0 bedeutet, daß der Lesevorgang erfolgreich beendet wurde; 1 zeigt das Ende der Datei an, es folgen keine Daten mehr (wenn die Datensätze gelesen wurden, ist der letzte Satz komplett). Probleme mit dem Diskettentransfersegment sind aufgetaucht, wenn in AL der Wert 2 zu finden ist (es handelt sich zumeist um ein "Umschlagen" des Segment-Offsets über hex FFFF, was nicht erlaubt ist). AL gleich 3 zeigt das Ende der Datei an, wobei der letzte Datensatz nicht vollständig gelesen wurde (er wird mit Nullen aufgefüllt).
Unabhängig vom Statuscode steht in CX die Anzahl der gelesenen Datensätze (einschließlich der teilweise gelesenen) und das FCB-Feld für wahlfreien Zugriff wird auf den nächsten Datensatz im wahlfreien Zugriff gerichtet.
Vergleichen Sie die Funktion auch mit Funktion 33, die nur einen einzelnen Datensatz wahlfrei liest.

16.1.37 Funktion 40 (hex 28): Datensätze wahlfrei schreiben

Im Unterschied zu Funktion 34 kann Funktion 40 nicht nur einen einzelnen, sondern mehrere Datensätze schreiben. Der Zugriff auf die Position des ersten Datensatzes erfolgt wahlfrei, alle anderen zu schreibenden Datensätze werden sequentiell angehängt. Der Zeiger DS:DX deutet auf den FCB der Datei, dem die wahlfreie Position entnommen wird. CX enthält die Anzahl der zu schreibenden Datensätze, wobei auch der Wert 0 erlaubt ist. Bei CX gleich 0 interpretiert DOS den spezifizierten Datensatz als den letzten der Datei, alle nachfolgenden Sätze werden abgeschnitten. Das erleichtert die Verwaltung von wahlfreien Dateien beträchtlich: Wenn Daten am Ende einer Datei logisch gelöscht werden, erlaubt diese Funktion, die Datei auch physikalisch auf die entsprechende neue Länge zurechtzustutzen, indem die neue Dateilänge in das Register CX geschrieben wird. Dadurch wird Speicherplatz auf der Diskette freigegeben, der sonst zwar belegt, aber logisch ungenutzt wäre.
Der Statuscode entspricht dem der Funktion 34: AL gleich 0 steht für eine erfolgreiche Durchführung. Der Wert 1 zeigt an, daß auf der Diskette nicht mehr genug Platz vorhanden ist. Unabhängig vom Statuscode enthält CX die Anzahl der geschriebenen (nicht der zu schreibenden) Datensätze.
Im Zusammenhang mit dieser Funktion sollten Sie sich die Funktion 34 näher ansehen, die nur einen einzigen Datensatz wahlfrei schreibt.

16.1.38 Funktion 41 (hex 29): Dateiname durchsuchen

Funktion 41 durchsucht eine Kommandozeile nach einem Dateinamen der Form *Laufwerk:Dateiname.Erweiterung*. Ist der Dateiname gefunden, wird ein FCB angelegt. Dabei ist die Doppelbedeutung des Wortes "Dateiname"

genau zu beachten. Im weiteren Sinne ist darunter die gesamte Dateispezifikation *Laufwerk:Dateiname.Erweiterung*, also einschließlich Laufwerksangabe und Namenserweiterung, zu verstehen. Im engeren Sinne ist der Dateiname nur der eigentliche Name ohne Laufwerksangabe und Namenserweiterung. Diese unglückliche Doppelverwendung hat sich leider mittlerweile weitgehed eingebürgert.
Funktion 41 ist besonders nützlich für die Verarbeitung von Dateinamenparametern, die einem Programm bei Aufruf übergeben werden sollen. Die Parameter werden in einzelne Teile aufgegliedert. Dadurch wird es für das Programm leichter, eigene Standardvorgaben bereitzustellen (Laufwerk, Dateiname oder Erweiterung), die bei der Kommandoeingabe überschrieben werden können. Diese Methode ist in der Praxis recht weit verbreitet.
Das Registerpaar DS:SI ist auf den Dateinamenstring, der alle drei genannten Elemente enthält, gerichtet. ES:DI zeigt auf den Speicherbereich, der mit einem ungeöffneten FCB gefüllt werden soll und die Bits 0 bis 3 im AL-Register kontrollieren die Aufsplittung des Dateinamens.
Ist Bit 0 gesetzt (1), werden Trennzeichen wie z.B. führende Leerzeichen bei der Suche nach dem Dateinamen übergangen. Bei Bit 0 gleich 0 hingegen erwartet die Funktion den Dateinamen direkt am Beginn der Kommandozeile.
Ist Bit 1 gleich 1, wird das Laufwerks-Byte im FCB nur gesetzt, wenn es im gesuchten Dateinamen spezifiziert ist. Das erlaubt es, im FCB, ein Laufwerk vorzugeben (das aber überschrieben werden kann), ohne auf DOS angewiesen zu sein.
Bei gesetztem Bit 2 wird der Dateiname im engeren Sinne (also ohne Laufwerksangabe und Namenserweiterung) im FCB nur dann geändert, wenn ein gültiger Dateiname in der Kommandozeile gefunden wird. Das gibt Programmen die Möglichkeit, einen Standard vorzugeben, der durch eine Kommandoeingabe überschrieben werden kann.
Steht Bit 3 auf 1, wird die Erweiterung des Dateinamens nur geändert, wenn ein gültiger Dateiname gefunden wird.
Wenn die Zerlegung des Dateinamens beendet ist, wird die normale Notation interpretiert. Das Dateigruppenzeichen "*" wird dabei in das präzisere "?" umgewandelt.
Wie gewöhnlich enthält AL das Ergebnis der Funktion: AL gleich 0 zeigt eine gelungene Durchführung mit einem einzelnen Dateinamen an; AL gleich 1 signalisiert den Erfolg der Operation mit einem Dateigruppenzeichen ("*" oder "?"), was im allgemeinen zur "Mehrfach-Such-Verarbeitung" zwingt (siehe Funktionen 17 und 18). AL gleich 255 (hex FF) zeigt einen Mißerfolg an (generelle Schwierigkeiten im Dateinamen im weiteren Sinne).
Um die wiederholte Durchführung zu erleichtern, wird DS:SI (eigentlich nur SI) jeweils auf den neuesten Stand gebracht, so daß der Zeiger auf das Zeichen nach dem Dateinamen (im weiteren Sinne) deutet. Wenn die

Suche erfolglos war, ist das zweite Byte des FCB (ES:DI+1) ein Leerzeichen.
Da die Routine eine traditionelle DOS-Routine (und keine erweiterte) ist, kann sie keine Pfadnamen verarbeiten. Dennoch ist die Routine nützlich.

16.1.39 Funktion 42 (hex 2A): Datum ablesen

Funktion 42 holt das aktuelle Datum aus dem entsprechenden DOS-Eintrag in die Register CX und DX. DH enthält den Monat (1 bis 12), DL den Tag (1 bis 28, 29, 30 oder 31, je nach Monat) und CX das Jahr (1980 bis 2099). Der Wochentag findet sich in Register AL als Wert von 0 bis 6 (Sonntag bis Samstag). Die Wochentagsfunktion scheint weitgehend unbekannt zu sein, obgleich sie recht nützlich sein kann.
Wie Sie die Routine einsetzen können, ist dem Beispiel in Kapitel 16.3 zu entnehmen.

16.1.40 Funktion 43 (hex 2B): Datum stellen

Funktion 43 setzt das Datum in den Registern CX und DX. Das Datumsformat entspricht dem von Funktion 32 verwendeten Format: DH enthält den Monat (1 bis 12), DL den Tag des Monats (1 bis 28, 29, 30 oder 31), CX das Jahr (1980 bis 2099) und AL den Wochentag (0 bis 6 von Sonntag bis Samstag).
Unter Funktion 42 finden Sie weitere Informationen hierzu. Blättern Sie zu Kapitel 16.3, dort finden Sie ein Anwendungsbeispiel.

16.1.41 Funktion 44 (hex 2C): Tageszeit ablesen

Funktion 44 meldet die Tageszeit. Die Zeit wird aus dem Zählerstand des ROM-BIOS errechnet, der sich aus dem Systemtakt ableitet (siehe Kapitel 12.5.6). DOS berücksichtigt das Mitternachtssignal des ROM-BIOS und setzt das Datum alle 24 Stunden auf den nächsten Tag weiter.
Der Zählerstand wird in eine Zeitangabe umgewandelt und in die Register CX und DX gespeichert. CH enthält die Stunde (0 bis 23 bei einer 24-Stunden-Uhr), CL die Minuten (0 bis 59), DH die Sekunden (0 bis 59) und DL die hundertstel Sekunden (0 bis 99).
Da der Systemtakt ungefähr 0,054 Sekunden beträgt, kann die Zeit nicht auf hundertstel Sekunden genau gemessen werden. Die maximale Genauigkeit liegt bei ca. 1/20 Sekunde. Dennoch werden alle Einzelwerte zwischen 0 und 99 durch den Algorithmus, den DOS verwendet, gemeldet. Immerhin lassen sich die hundertstel Sekunden daher als Initialisierungswert eines Pseudozufallszahlengenerators verwenden.
Der Wochentag wird nicht durch Funktion 44, sondern durch Funktion 42 gemeldet, auch wenn hier und da etwas anderes zu lesen ist.

16.1.42 Funktion 45 (hex 2D): Tageszeit stellen

Funktion 45 setzt die Tageszeit, die in den Registern CX und DX spezifiziert wird. CH enthält die Stunden (0 bis 23 für eine 24-Stunden-Uhr), CL die Minuten (0 bis 59), DH die Sekunden (0 bis 59) und DL die hundertstel Sekunden (0 bis 99).

16.1.43 Funktion 46 (hex 2E): Diskettenschreibverifikation setzen

Funktion 46 bestimmt, ob jede Diskettenschreiboperation unmittelbar nach dem Schreibvorgang überprüft werden soll oder nicht. Ist die Verifikation aktiv, werden die geschriebenen Daten nochmals gelesen, um sie auf Korrektheit zu überprüfen. Das bedeutet nicht, daß der Inhalt des Geschriebenen verglichen wird, vielmehr wird lediglich ein Paritätstest (CRC-Test) ausgeführt.
Wenn Sie die Routine verwenden wollen, muß in DL der Wert 0 stehen. Zukünftige DOS-Versionen werden unter Umständen auch andere Werte akzeptieren. AL wird auf 1 oder 0 gesetzt, je nachdem, ob die Verifikation aktiv sein soll (1) oder nicht (0).
Ab DOS-Version 2.00 kann die erweiterte Funktion 84 (hex 54) verwendet werden, um den Status (Verifikation aktiv oder nicht) zu erfahren. Sehen Sie hierzu in Kapitel 17 nach.

16.2 Der Dateikontrollblock (FCB)

Die traditionellen DOS-Funktionsaufrufe arbeiten mit einem zentralen *DOS-Dateikontrollblock* (*File Control Block* oder FCB), um den herum die Dateien angelegt werden. Der FCB besteht aus 44 Bytes, die Informationen über die zugehörige Datei enthalten. Der FCB hat den großen Vorteil, sehr leicht manipulierbar zu sein, wie die Erklärungen zu vielen Funktionsaufrufen beweisen. Sie werden sehen, daß die neuen DOS-Funktionen (sie werden in Kapitel 17 behandelt), die mit der Version 2.00 eingeführt wurden, im Gegensatz zum FCB-Konzept einen Großteil der Dateikontrollinformation verborgen halten. Statt mit FCBs wird dort mit logischen Dateinummern gearbeitet, was zwar wesentlich einfacher ist, aber keinen so leichten Eingriff in die internen Vorgänge erlaubt. Das wird im nächsten Kapitel ausführlich erläutert.
Es gibt zwei Hauptteile im FCB: den FCB selbst, einen 37-byte-Bereich und den erweiterten FCB, einen Vorspann des FCB mit einer Länge von 7 Bytes. Der größere Teil des FCB speichert Kontrolldaten über die Datei, wie z.B. den Dateinamen, die Laufwerksspezifikation, die Datensatzlänge und die Anzahl der Datensätze. Die 7-byte-Erweiterung speichert bestimmte Dateiattribute (in Kapitel 5.5.2.3 finden Sie mehr über Dateiattribute).

Die 7-byte-FCB-Erweiterung wird nur gebraucht, wenn Dateien mit ungewöhnlichen Attributen wie z.B. *versteckt* oder Systemattributen versehen sind. Unter normalen Umständen wird die Erweiterung nicht benötigt, weshalb soll man also die 7 Byte Speicherplatz dafür reservieren? Aber es kommt noch "besser": Der Wert 255 (hex FF) im ersten Byte der FCB-Erweiterung signalisiert, daß die Erweiterung vorhanden ist. Das kann zu der absurden Situation führen, daß sieben Bytes reserviert werden, die die einzige Aufgabe haben, mitzuteilen, daß sie überflüssig sind. Die DOS-Entwickler sind eben auch nur Menschen.
Der FCB wird vom Beginn des Hauptteils an adressiert, die Felder im FCB werden nach ihrem Offset von dieser Adresse ab unterschieden. Das bedeutet, daß die Erweiterung einen negativen Offset von -7 erhält, da sie vor dem Hauptteil liegt.

Offset (Dez)	Größe (Byte)	Verwendung	Beschreibung
−7	1	frei (1)	Erweiterung aktiv? FF = Ja, ansonsten nein
−6	5	frei (1)	Unbenutzt: sollte auf 0 gesetzt werden
−1	1	frei (1)	Dateiattribut, wenn Erweiterung aktiv
0	1	frei (1)	Spezielle Laufwerksnummer
1	8	frei (1)	Datei- oder Gerätename
9	3	frei (1)	Dateinamenerweiterung
12	2	frei (2)	Aktuelle Blocknummer
14	2	frei (2)	Datensatzlänge
16	4	DOS	Dateilänge in Bytes
20	2	DOS	Dateidaten (bitweise kodiert, wie im Dateiverzeichnis)
22	10	DOS	DOS-Kontrollarbeitsbereich
32	1	frei (2)	Aktuelle Datensatznummer (von 0 bis 127, siehe Text)
33	4	frei (2)	Datensatznummer für wahlfreien Zugriff

Tabelle 16-3 Die Felder des Dateikontrollblockes

Die von DOS verwalteten Felder sollten nicht verändert werden. Die anderen Felder können mit folgendem Vorbehalt manipuliert werden: Felder, die mit *frei(1)* bezeichnet sind, sollten belegt werden, bevor die Datei geöffnet wird, die anderen (*frei(2)*), bevor Daten geschrieben oder gelesen werden.
Offsets -7 und -6: Das Signal, daß die Erweiterung aktiv ist, besteht darin, daß das erste Byte (Offset -7) den Wert 255 (hex FF) enthält. Bei jedem anderen Wert ist die Erweiterung nicht aktiv. Das darauf folgende Feld von fünf Bytes Länge sollte auf Null gesetzt werden, es kann dann auf keinen Fall stören.
Offset -1: Das Dateiattributfeld muß die speziellen Attribute der geöffneten Datei enthalten. Es wird für normale Dateien mit einem Attribut von 0 nicht benötigt. Die Attribute, die hier spezifiziert werden müssen

sind: *versteckt*, System- und Verzeichnisattribute. Die Attributspezifikation wird nicht benötigt, wenn auf eine Datei nur lesend zugegriffen werden soll.
Mehr über die Attributkodierung finden Sie in Kapitel 5.5.2.3.
Offset 0: Die spezielle Laufwerksnummer in diesem Byte wird verwendet, um das Laufwerk zu bestimmen, mit dem gearbeitet werden soll. Es werden nicht die normalen Laufwerksnummern verwendet (0 gleich Laufwerk A, 1 gleich Laufwerk B usw.), sondern ein flexibleres Format: Laufwerk A wird durch die Zahl 1 gekennzeichnet, Laufwerk B durch 2 usw. Die größere Flexibilität liegt darin, daß die Zahl 0 für das Standardlaufwerk zur Verfügung steht, egal, um welches Laufwerk es sich dabei handelt. Ist die Datei geöffnet, ändert DOS die Null in die entsprechende Ziffer, die mit der Funktion 25 (hex 19) eingesehen werden kann. Wenn Sie mit dem Feld arbeiten, sollten Sie aufpassen, das keine Verwirrung wegen der Numerierung der Laufwerke entsteht. Die hier verwendeten Ziffern sind um eins höher als die normalerweise benutzten.
Offsets 1 und 9: Die beiden Felder, die bei den Offsets 1 und 9 beginnen, enthalten den Dateinamen (im engeren Sinne) und die Dateinamenerweiterung. Nach den DOS-Konventionen sind die beiden Felder linksbündig und werden, soweit nötig, auf der rechten Seite mit Leerzeichen (CHR$(32), hex 20) aufgefüllt. Es können Groß- oder Kleinbuchstaben verwendet werden. Wenn der Dateiname ein Gerätename ist, den DOS erkennt, wie z.B. CON:, AUX:, COM1:, COM2:, LPT1:, LPT2:, PRN: oder NUL:, verwendet DOS das Gerät statt der Datei.
Es sei nochmals darauf hingewiesen, daß der FCB keine Pfadnamen verarbeiten kann. Ein Zugriff über den FCB bezieht sich immer auf das aktuelle Verzeichnis des jeweiligen Laufwerkes. Pfade und Unterverzeichnisse werden in den erweiterten Funktionen (siehe Kapitel 17) berücksichtigt.
Offsets 12 und 32: Bei sequentiellen Operationen dienen der momentane Block und die Datensatzfelder dazu, die Position in der Datei zu verfolgen. Die Verwendung der Felder ist ziemlich merkwürdig. Statt daß es eine integrierte Datensatznummer gibt, wird sie in zwei Teile aufgespalten. Der höhere Teil wird als *Blocknummer* bezeichnet, der niedere als *Datensatznummer*. Um die Sache noch komplizierter zu machen, besteht ein Block aus 128 Datensätzen, nicht aus 256, was die Speicherung als eine 1-byte-Nummer ermöglichen würde. Es sind also nur Werte zwischen 0 und 127 erlaubt.
Der erste Datensatz einer Datei trägt die Datensatznummer 0 in Block 0. Das Format wurde gewählt, weil es die schnelle Berechnung des nächsten Datensatzes für einen sequentiellen Zugriff, gleich welcher Art (Lesen oder Schreiben), ermöglicht. Mit diesem System kann DOS den sequentiellen Zugriff als Abwandlung des wahlfreien Zugriffs abarbeiten. Die Felder müssen vor der ersten sequentiellen Operation belegt werden. DOS führt die Numerierung nach jeder Operation weiter, was für den Pro-

grammierer natürlich sehr bequem ist. Durch Manipulation der Felder kann auf beliebige Datensätze zugegriffen werden, was im Zusammenhang mit Eingabedateien interessant sein kann.

Offset 14: Das Feld ab Offset 14 enthält die Länge der logischen Datensätze einer Datei in Byte. Wenn ein Programm über eine DOS Routine einen Datensatz liest oder schreibt, ist die logische Datensatzlänge gleich der Anzahl der Bytes, die zwischen dem DOS-Diskettenpuffer und dem Programmdatenbereich übertragen werden. Der Wert hat nichts mit der Datei auf der Diskette oder mit der Datei, wie sie von DOS erkannt wird zu tun, er zeigt nur an, wie das Programm die Daten behandelt.

Die Datensatzlänge ist nicht unbedingt ein Bestandteil einer Datei, sondern wird durch den Zugriff bestimmt. Solange eine Datei nicht mit festgelegten Datensatzlängen arbeitet, können wir die Datei so behandeln, als sei sie aus 1-byte-Einträgen aufgebaut, um Platz zu sparen. Es ist bei Texteditoren üblich, mit einer Datensatzlänge von 128 Bytes für ASCII-Dateien zu arbeiten, weil dadurch die Anzahl der nötigen Funktionsaufrufe reduziert wird. Beachten Sie aber: Wenn eine Datei mit Funktionsaufruf 15 geöffnet wird, legt DOS automatisch eine Datensatzlänge von 128 Byte fest. Jede andere Länge muß spezifiziert werden, nachdem die Datei geöffnet wurde.

Offset 16: Das Feld enthält die Dateilänge in Byte. Der Wert wird dem Verzeichniseintrag entnommen und in den FCB kopiert, wenn DOS eine Datei öffnet. Bei einer Ausgabedatei ändert DOS das Feld dynamisch mit dem Aufbau der Datei, so daß die Dateilängenangabe zu jedem Zeitpunkt der tatsächlichen Dateilänge entspricht. Wird die Datei geschlossen, erhält der Verzeichniseintrag den neuen Wert. Wenn Sie versuchen, ein Unterverzeichnis als Datei zu lesen (was die Benutzung der FCB-Erweiterung mit sich bringt), um das Dateiattribut des Verzeichnisses zu setzen, wird das Feld zu Null, sobald die Datei geöffnet wird, da der Verzeichniseintrag eines Unterverzeichnisses eine Länge von Null hat. Um ein Unterverzeichnis zu lesen, müssen Sie dieses Feld auf einen willkürlich gewählten, hohen Wert setzen. Und noch etwas ist zu beachten: Wenn Sie die Funktion zum Umbenennen von Dateien, Funktion 23, verwenden, wird der neue Name ab Offset 16 abgelegt, überschreibt also das Byte-Längenfeld.

Offset 20: Das Datumsfeld ist als 2-byte-Wort kodiert und benutzt dieselbe Form wie die Verzeichniseinträge: "MM/TT/JJ". Sobald eine Datei geöffnet wird, kopiert DOS das Datum aus dem Dateiverzeichnis in das Datumsfeld. In Kapitel 5.5.2.6 finden Sie mehr über die Kodierung.

Offset 33: Das Feld für wahlfreien Zugriff fungiert bei direkten oder wahlfreien Schreib- oder Leseoperationen als Speicher für die Datensatz- und Blocknummern. Das Feld besitzt die Form einer 4-byte-Ganzzahl oder 32-bit-Ganzzahl, die in Worte oder Bytes aufgeteilt werden kann. Die Numerierung der Datensätze beginnt bei 0. Es ist also leicht, den Datei-Offset jedes beliebigen Datensatzes duch Multiplikation der wahl-

freien Datensatznummer mit der Datensatzlänge zu errechnen. Das Feld muß vor jedem wahlfreien Zugriff gesetzt werden. DOS verändert es nicht.

16.3 Beispiel

Die folgende Routine aus den *Norton Utilities* dient zur Berechnung des Wochentages aus einem Datum. Der Zeitraum, der der Berechnung zugrunde gelegt wird, beginnt mit dem 1. Januar 1980, einem Dienstag, und endet am Donnerstag, den 31. Dezember 2099.
Es gibt viele interessante und ausgeklügelte Algorithmen, um den Wochentag zu bestimmen. Ab Version 2.00 ist in DOS eine Routine eingebaut, die den Wochentag des aktuellen Datums berechnet. Indem man als aktuelles Datum das interessierende Datum angibt, kann man zu jedem beliebigen Datum den Wochentag herausfinden. Das folgende Beispiel zeigt Ihnen, wie das funktioniert.
Das Assemblerprogramm ist nicht nur sehr raffiniert programmiert, sondern zeigt auch, wie drei DOS-Funktionsaufrufe zusammenarbeiten, um ein Resultat zu erzeugen. Außerdem verdeutlicht es die kleinen Schwierigkeiten, die bei der Benutzung des Stapels anfallen, wenn Daten gesichert und wieder zurückgespeichert werden. Wie Sie sehen werden, muß die Verwaltung des Stapels sehr sorgfältig ausgeführt werden, damit sich nicht verschiedene Werte in die Quere kommen.
Die Unterroutine, die WOCHENTAG heißt, ist so geschrieben, daß sie mit dem Lattice/Microsoft C Compiler genutzt werden kann. Die Routine wird mit drei Ganzzahlvariablen aufgerufen, die Monat, Tag und Jahr des Datums angeben, von dem wir den Wochentag wissen wollen. Die Routine gibt den Wochentag als Zahl von 0 bis 6 (Sonntag bis Samstag) zurück. Das fügt sich sehr günstig in die Konventionen der Sprache C über Variablenfelder ein: Ein Stringfeld, das die ausgeschriebenen Wochentage enthält, kann über Indizes adressiert werden. Deshalb verwenden wir die Unterroutine folgendermaßen:

```
TAG_NAME (WOCHENTAG (MONAT,TAG,JAHR))
```

Es ist wichtig zu wissen, daß die Routine "blind" mit den Daten arbeitet, sie werden weder auf Gültigkeit geprüft noch wird getestet, ob der Arbeitsbereich von DOS eingehalten wird. Und noch etwas: Die Routine arbeitet nur ab DOS-Version 2.00. Hier die Unterroutine:

```
          PGROUP  GROUP PROG
          PUBLIC  WOCHENTAG
PROG      SEGMENT BYTE PUBLIC 'PROG'
          ASSUME  CS:WOCHENTAGE
WOCHENTAG PROC    NEAR
;Strategie: bisheriges Datum sichern
;neues Datum setzen
;zugehörigen Wochentag ablesen
;altes Datum zurückspeichern
          PUSH    BP
          MOV     BP,SP
          MOV     AH,2AH      ;Datum zum Sichern nehmen
          INT     21H         ;DOS-Funktionsaufruf
          PUSH    CX          ;altes Datum...
          PUSH    DX          ;... im Stapel ablegen
          MOV     CX,[BP+8]   ;Jahr
          MOV     DL,[BP+6]   ;Tag
          MOV     DH,[BP+4]   ;Monat
          MOV     AH,2BH      ;Datum umsetzen
          INT     21H         ;DOS-Funktionsaufruf
          MOV     AH,2AH      ;Wochentag holen
          INT     21H         ;DOS-Funktionsaufruf
          POP     DX          ;altes Datum ...
          POP     CX          ;... aus dem Stapel zurückholen
          Push    AX          ;Wochentag im Stapel sichern
          MOV     AH,2BH      ;altes Datum zurückspeichern
          INT     21H         ;DOS-Funktionsaufruf
          POP     AX          ;Wochentag aus Stapel holen
          MOV     AH,0        ;höherwertigen Teil löschen
          POP     BP
          RET
          WOCHENTAG ENDP
          PROG      ENDS
          END
```

Kapitel 17
Neue DOS-Funktionen

17.1 DOS-2-Erweiterungen 279
17.1.1 Vorteile der erweiterten DOS-Funktionen 279
17.1.2 Standardfehlercodes 280
17.1.3 ASCIIZ-Strings 281
17.1.4 Dateinummern 281
17.1.5 Installierbare Schnittstellentreiber 282
17.2 Die erweiterten DOS-Funktionen 283
17.2.1 Funktion 47: DTA-Adresse feststellen 283
17.2.2 Funktion 48: DOS-Version feststellen 283
17.2.3 Funktion 49: Erweitertes Beenden und im Speicher verbleiben (KEEP) 285
17.2.4 Funktion 51 : Ctrl-Break-Abfrage testen oder festlegen 285
17.2.5 Funktion 53 : Interruptvektor feststellen 286
17.2.6 Funktion 54 : Freie Diskettenkapazität feststellen 286
17.2.7 Funktion 56 : Landesabhängige Information lesen 287
17.2.8 Funktion 57 : Unterverzeichnis anlegen (MKDIR) 289
17.2.9 Funktion 58 : Unterverzeichnis löschen (RMDIR) 289
17.2.10 Funktion 59 : Aktuelles Verzeichnis ändern (CHDIR) 290
17.2.11 Funktion 60 : Datei anlegen (CREAT) 290
17.2.12 Funktion 61 : Datei öffnen 290
17.2.13 Funktion 62 : Datei schließen 292
17.2.14 Funktion 63 : Lesen (Datei oder Gerät) 292
17.2.15 Funktion 64: Schreiben (Datei oder Gerät) 292
17.2.16 Funktion 65: Datei löschen 293
17.2.17 Funktion 66: Dateizeiger bewegen 293
17.2.19 Funktion 67: Dateiattribute festlegen/feststellen (CHMOD) 294
17.2.19 Funktion 68 : Ein-/Ausgabesteuerung für Geräte (IOCTL) 294
17.2.20 Funktion 69 : Dateinummern duplizieren (DUP) 296
17.2.21 Funktion 70 : Dateinummernduplizierung erzwingen (CDUP) 296
17.2.22 Funktion 71 : Aktuelles Verzeichnis melden 297
17.2.23 Funktion 72 : Speicherbereich belegen 297

17.2.24 Funktion 73: Speicherbereich freigeben 298
17.2.25 Funktion 74: Größe des belegten Speicherbereichs verändern (SETBLOCK) 298
17.2.26 Funktion 75: Programm laden und ausführen (EXEC) 298
17.2.27 Funktion 76: Erweitertes Programm beenden 300
17.2.28 Funktion 77: Statuscode des Unterprogramms melden 300
17.2.29 Funktion 78: Dateisuche beginnen (FIND FIRST) 301
17.2.30 Funktion 79: Dateisuche fortsetzen 301
17.2.31 Funktion 84: Verifizierstatus feststellen 302
17.2.32 Funktion 86: Datei umbenennen 302
17.2.33 Funktion 87: Datum und Zeit für Datei stellen/ablesen 302
17.3 DOS-3-Erweiterungen 303
17.3.1 Funktion 89: Erweiterten Fehlercode melden 303
17.3.2 Funktion 90: Temporäre Datei anlegen 305
17.3.3 Funktion 91: Neue Datei anlegen 306
17.3.4 Funktion 92: Dateizugriff sperren/freigeben 306
17.3.5 Funktion 98: PSP-Adresse feststellen 307

Während wir uns im letzten Kapitel mit den traditionellen DOS-Funktionen, die in jeder DOS-Version vorhanden sind, beschäftigt haben, wollen wir uns in diesem Kapitel den neuen, erweiterten DOS-Funktionen, die mit der Version 2.00 eingeführt wurden, zuwenden. Die erweiterten Funktionen von DOS 2.10 und DOS 2.00 sind identisch. Ab Version 3.00 wurden einige der bereits existierenden Funktionen verändert und sechs neue hinzugefügt. Sie finden in diesem Kapitel alle neuen DOS-Funktionen ausführlich beschrieben, wobei die mit DOS 3.00 vorgestellten Funktionen am Ende des Kapitels zu finden sind. Bei jeder Funktion wird auf eventuelle Unterschiede zwischen DOS 2.00 bzw. 2.10 und DOS 3.00 hingewiesen.
In Kapitel 15 finden Sie die DOS-Interrupts, die ähnlich wie die DOS-Funktionen einsetzbar sind. In Kapitel 16 sind die traditionellen DOS-Funktionsaufrufe aufgelistet und erläutert. Kapitel 18 enthält eine tabellarische Zusammenfassung der DOS-Routinen.

17.1 DOS-2-Erweiterungen

Jede neue DOS-Version brachte Neuerungen mit sich, am drastischsten aber waren die zusammen mit DOS 2.00 vorgestellten Änderungen. Zu den 42 existierenden Funktionen kamen 33 neue hinzu. DOS 2.00 ermöglicht unter anderem einen leichteren Dateizugriff und die Unterstützung praktisch jedes Hardwarezusatzes über *installierbare Schnittstellentreiber*.

17.1.1 Vorteile der erweiterten DOS-Funktionen

Viele der mit DOS 2.00 und 3.00 neu vorgestellten DOS-Routinen bringen drei wichtige Eigenschaften mit sich, die unabhängig von der Art der Funktionen Einfluß auf die Programmierung haben. Erstens melden die meisten neuen Funktionen Standardfehlercodes in das Register AX. Zweitens verwenden Funktionen, die mit Zeichenketten arbeiten, ein spezielles Stringformat, das ASCIIZ-Format. Eine ASCIIZ-Zeichenkette wird durch ein Null-Byte beendet ("Z" steht für "Zero" oder Null). Drittens arbeiten viele neue Funktionen mit einer 16-bit-Nummer, die *Dateinummer* (*File-Handle*) genannt wird. Die Dateinummern werden statt der Dateikontrollblöcke (FCBs) verwendet, um Dateien und Ein- und Ausgabeprozesse zu steuern und zu überwachen. Alle drei Neuerungen, die sich vorteilhaft auf die Arbeit mit DOS auswirken, werden im folgenden ausführlich beschrieben.

17.1.2 Standardfehlercodes

Viele der erweiterten DOS-Funktionen melden einen Fehlercode in das AX-Register. Aus dem Code läßt sich ablesen, ob die Funktion korrekt ausgeführt wurde oder nicht und gegebenenfalls welcher Fehler auftrat. Statt von *Fehlercodes* spricht man auch von *Statuscodes*.
Die Standardstatuscodes werden in das AX-Register gemeldet, obwohl ein Halbregister völlig ausreichen würde, da alle Codewerte unter 255 liegen. Die Carry-Flagge (CF) sollte anzeigen, ob ein Fehler vorliegt (CF gleich 1) oder nicht, erfüllt diese Funktion jedoch leider nicht in jedem Fall. Manchmal wird CF unter Bedingungen gesetzt, die man kaum als Fehler bezeichnen kann. Glücklicherweise sind aber die Standardfehlercodes verläßlich. Es empfiehlt sich, das AX-Register zur Statusprüfung zu verwenden und die Carry-Flagge außer acht zu lassen. Bei der Beschreibung der Funktionen, die einen Fehlercode zurückgeben, finden Sie eine kurze Liste der Codes, die meist korrekt durch die CF-Flagge angezeigt werden.

Hinweis: DOS 3.00 offeriert nicht nur die Standardfehlercodes, sondern zusätzliche Fehlercodes durch Funktion 89. Lesen Sie hierzu Kapitel 17.3.1 durch.

Fehlercode Dez	Hex	Bedeutung
1	1	Ungültige Funktionsnummer
2	2	Datei nicht gefunden
3	3	Pfad nicht gefunden
4	4	Keine Dateinummer frei verfügbar
5	5	Zugriff nicht möglich
6	6	Ungültige Dateinummer
7	7	Speicherkontrollblöcke zerstört
8	8	Unzureichende Speicherkapazität
9	9	Ungültige Speicherblockadresse
10	A	Ungültige Umgebung (siehe DOS-Kommando SET)
11	B	Ungültiges Format
12	C	Ungültiger Zugriffscode
13	D	Ungültige Daten
14	E	Unbenutzt
15	F	Ungültige Laufwerksspezifikation
16	10	Versuch, Standardverzeichnis zu verlagern
17	11	Anderes Gerät
18	12	Keine weiteren Datein gefunden

Tabelle 17-1 Die Standardfehlercodes, die in Register AX übergeben werden, wenn ein Funktionsaufruf erfolglos verlaufen ist

17.1.3 ASCIIZ-Strings

Wenn man mit Zeichenketten variabler Länge arbeitet, muß man das Stringende mit einem Sonderzeichen kennzeichnen. Ab DOS 2.00 stehen ASCIIZ-Strings zur Verfügung. Das letzte Zeichen eines ASCIIZ-Strings ist ein Null-Byte (das Z steht für "Zero", null). Die Zeichenkette selbst besteht aus normalen ASCII-Zeichen. Ein gutes Beispiel für eine Zeichenkette mit variabler Länge ist ein Pfadname, je nach Pfadlänge und Namen ist er kürzer oder länger. Ein typischer String, der einen Pfadnamen darstellt, hat folgendes Aussehen:

```
A:\VERZEICHNIS1\VERZEICHNIS2\DATEINAME.ERW(NULL-BYTE)
```

Der Laufwerksbuchstabe (A:) kann weggelassen werden. Als Trennzeichen zwischen den einzelnen Namen sind Schrägstrich (/) und umgekehrter Schrägstrich (\) gleichermaßen erlaubt.
Der ASCIIZ-String wird sowohl im UNIX-Betriebssystem als auch in der Programmiersprache C verwendet. DOS 2.00 zeigt generell eine Orientierung in Richtung UNIX und C.

17.1.4 Dateinummern

Die traditionellen DOS-Funktionen wickeln die Dateiverwaltung mit Dateikontrollblöcken (FCBs) ab und erlauben dem Programmierer über die FCBs sehr weitgehende Eingriffe in das Dateimanagement. Mit der Einführung der Dateinummern mit DOS 2.00 hat sich das grundlegend geändert. Der Programmierer arbeitet fast ausschließlich mit den Dateinummern und erhält kaum noch Zugang zu den internen Dateiinformationen.
Bei den traditionellen Funktionen spezifizieren wir die Datei, mit der gearbeitet werden soll, indem ein Zeiger auf den FCB der Datei gerichtet wird. Für das Dateimanagement stehen diverse Routinen zur Verfügung, die überwiegend über den FCB gesteuert werden. Bei den erweiterten Funktionen dagegen, arbeiten wir mit einer Dateinummer im AX-Register, die nichts weiter als eine 16-bit-Zahl ist. Die Dateinummern bezeichnen nicht nur die Dateien und Geräte, mit denen gearbeitet wird, sondern steuern auch den größten Teil der Dateiverwaltung.
Der Programmierer erhält die Kontrolle über die Dateinumnern nur zum Anlegen oder Öffnen einer Datei, ansonsten liegt die Dateinummernverwaltung bei DOS. Es gibt fünf Standarddateinummern (0 bis 4), die jedem Programm zur Verfügung stehen. Weitere Dateinummern werden von DOS zugeteilt, sofern sie benötigt werden.

Die traditionellen DOS-Funktionen erlauben es, mit dem DOS-Kommando FILES maximal 99 Dateien zu öffnen und gleichzeitig zu verwalten. Der AT kann mit bis zu 255 offenen Dateien arbeiten. Die erweiterten Funktionen von DOS 2.00 beschränken die maximale Anzahl von Dateinummern (das heißt, gleichzeitig offenen Dateien) für jedes Programm auf 20.

Dateinummer	Verwendung	Standard
0	Standardeingabe (normalerweise Tastatureingabe)	CON:
1	Standardausgabe (normalerweise Bildschirmausgabe)	CON:
2	Standardfehlerausgabe (immer Bildschirmausgabe)	CON:
3	Standardhilfsgerät (AUX:)	AUX:
4	Standarddrucker (LPT1: oder PRN:)	PRN:

Tabelle 17-2 Die fünf Standarddateinummern, die jeder Datei zugänglich sind

In DOS 3.00 lassen sich einem Programm auch mehr als 20 Dateinummern zuordnen, der Standard ist jedoch auch hier 20. Diese Einschränkungen sind nur für sehr komplexe Programme von Bedeutung.

17.1.5 Installierbare Schnittstellentreiber

DOS 2.00 brachte als Novum die Möglichkeit mit sich, Treibersoftware für verschiedene Geräte in DOS integrieren zu können. Man spricht von *installierbaren Geräte-* oder *Schnittstellentreibern.* Während die meisten Treiberprogramme zur entsprechenden Hardware, z.B. einem Drucker, gehören, erhalten Sie den Standard-ANSI-Treiber, meist ANSI.SYS genannt, mit DOS zusammen.
Die DOS-Schnittstellentreiber haben die Aufgabe, Hardware, die an das Computersystem angeschlossen wird, zu unterstützen. Es gibt Treiber, die zu einem speziellen Gerät gehören (z.B. einer Maus oder einem Joystick), und andere, die die Funktionsweise der Standardhardware verändern. Zur letztgenannten Art zählt der ANSI-Treiber, der die Tastatur und den Bildschirm unterstützt. Mit Treiberprogrammen kann man Funktionen implementieren, die im BIOS und in DOS nicht vorhanden sind. Indem die Schnittstellentreiber als Teil des DOS installiert werden, vermeidet man, auf BIOS- oder gar auf der Hardwareebene zu programmieren, was ohne Treibersoftware erforderlich wäre, um neue Geräte anzupassen. Das ist ein wichtiger Aspekt bei Programmen, die in Fenster- oder Multitasking-Umgebungen ablaufen sollen.
Das Konzept der Schnittstellentreiber wurde zwar mit DOS 2.00 eingeführt, gehört aber nicht zu den erweiterten DOS-Funktionen. Daher wird im folgenden nicht weiter auf die Treiber eingegangen. Sie werden in Anhang A separat behandelt. Das bedeutet aber nicht, daß sie unwichtig sind. Im Gegenteil! Für einen professionellen Programmierer ist es unerläßlich, sich mit der Treiberkonzeption vertraut zu machen.

17.2 Die erweiterten DOS-Funktionen

Die mit DOS 2.00 neu aufgenommenen DOS-Funktionsaufrufe werden über Interrupt 33 (hex 21) angesprochen. Die einzelnen Funktionen werden durch eine Funktionsnummer in Register AH ausgewählt. Wenn in einem Programm eine neue DOS-Funktion aufgerufen werden soll, empfiehlt es sich, zuvor die DOS-Versionsnummer abzufragen und damit sicherzustellen, daß die Funktion verfügbar ist.

Funktion Dez	Hex	Einteilung
47–56	2F–38	Erweiterte Funktionen
57–59	39–3B	Verzeichnis
60–70	3C–46	Erweiterte Dateiverwaltung
71	47	Verzeichnis
72–75	48–4B	Erweiterte Speicherverwaltung
76–79	4C–4F	Erweiterte Funktionen
84–87	54–57	Erweiterte Funktionen
88–98	58–62	DOS 3.00-Erweiterungen

Tabelle 17-3 Die Einteilung der erweiterten DOS-Funktionen

Die Einteilung der neuen DOS-Funktionen in logisch zusammengehörige Gruppen, wie sie in diesem Buch geschieht, weicht etwas von der üblichen Einteilung ab. Bei Vergleichen mit anderen Publikationen sollten Sie darauf achten, um Mißverständnisse zu vermeiden.

17.2.1 Funktion 47 (hex 2F): DTA-Adresse feststellen

Nach Aufruf der Funktion steht die Adresse des momentan von DOS verwendeten Diskettentransferbereiches (DTA) im Registerpaar ES:BX. Beachten Sie, daß hier das ES-Register für den Segmentteil verwendet wird, es gibt also keine Probleme mit dem Programmablauf. Vergleichen Sie das bitte mit Funktion 26 und den Erläuterungen über das DS-Register zu Funktion 27 in Kapitel 16.

17.2.2 Funktion 48 (hex 30): DOS-Version feststellen

Funktion 48 ermittelt die DOS-Versionsnummer in zwei Teilen. Der erste Teil wird in Register AL (für DOS 2.10 wäre das die 2), der zweite Teil in Register AH übergeben (für DOS 2.10 wäre das 10). Erfolgt der Aufruf der Routine in DOS 1, wo die Funktion gar nicht implementiert ist,

Funktion Dez	Funktion Hex	Beschreibung	Funktion Dez	Funktion Hex	Beschreibung
47	2F	DTA-Adresse feststellen	72	48	Speicherbereich belegen
48	30	DOS-Version feststellen	73	49	Speicherbereich freigeben
49	31	Erweitertes Beenden und im Speicher verbleiben (KEEP)	74	4A	Größe des belegten Speicherbereichs verändern (SETBLOCK)
50	32	Unbenutzt	75	4B	Programm laden und ausführen (EXEC)
51	33	Ctrl-Break-Abfrage testen oder festlegen	76	4C	Erweitertes Programm beenden
52	34	Unbenutzt	77	4D	Statuscode des Unterprogramms melden
53	35	Interruptvektor feststellen	78	4E	Dateisuche beginnen (FIND FIRST)
54	36	Freie Diskettenkapazität feststellen	79	4F	Dateisuche weiterführen
55	37	Unbenutzt	80	50	Unbenutzt
56	38	Landesabhängige Information lesen	81	51	Unbenutzt
57	39	Unterverzeichnis anlegen (MKDIR)	82	52	Unbenutzt
58	3A	Unterverzeichnis löschen (RMDIR)	83	53	Unbenutzt
59	3B	Aktuelles Verzeichnis ändern (CHDIR)	84	54	Verifizierstatus feststellen
60	3C	Datei anlegen (CREAT)	85	55	Unbenutzt
61	3D	Datei öffnen	86	56	Datei umbenennen
62	3E	Datei schließen	87	57	Datum und Zeit für Datei stellen/ablesen
63	3F	Lesen (Datei oder Gerät)	88	58	Unbenutzt
64	40	Schreiben (Datei oder Gerät)	89	59	Erweiterten Fehlercode melden
65	41	Datei löschen	90	5A	Temporäre Datei anlegen
66	42	Dateizeiger bewegen	91	5B	Neue Datei anlegen
67	43	Dateiattribute festlegen/feststellen (CHMOD)	92	5C	Dateizugriff sperren/freigeben
68	44	Ein-/Ausgabesteuerung für Geräte (IOCTL)	93	5D	Unbenutzt
69	45	Dateinummern duplizieren (DUP)	94	5E	Unbenutzt
70	46	Dateinummernduplizierung erzwingen (CDUP)	95	5F	Unbenutzt
71	47	Aktuelles Verzeichnis melden	96	60	Unbenutzt
			97	61	Unbenutzt
			98	62	PSP-Adresse feststellen

Tabelle 17-4 Die erweiterten DOS-Funktionen; Aufruf mit Interrupt 33, Auswahl in Register AH

wird in AL eine 0 gemeldet, der Wert in AH ist bedeutungslos. In einem Programm, in dem die neuen Funktionen verwendet werden, sollte vor dem ersten Aufruf einer neuen Routine mit Funktion 48 getestet werden, welche DOS-Version vorliegt. Wird als Ergebnis eine Versionsnummer unter 2 bzw. unter 3 gemeldet, können keine DOS-2 bzw. DOS-3-Funktionen aufgerufen werden.

17.2.3 Funktion 49 (hex 31): - Erweitertes Beenden und im Speicher verbleiben (KEEP)

Funktion 49 ist die erweiterte Version der mit dem DOS-Interrupt 39 aufzurufenden Routine *Beenden und im Speicher verbleiben.* Die Funktion beendet nicht nur den Programmablauf, sondern übermittelt zusätzlich einen Wert des Programmes in das Register AL. Der Wert kann mit ERRORLEVEL im DOS-Stapelverarbeitungsbetrieb (Batch-Betrieb) ausgewertet werden.
Bei einem Programm, das mit Funktion 49 beendet wird, muß DOS die Information erhalten, welcher Programmteil im Speicher verbleiben und welcher Teil gelöscht werden soll. Dazu wird in DX *die* Segmentadresse geschrieben, die auf das erste zu löschende Speichersegment zeigt.

Hinweis: Ein Programm kann mit den Speicherbelegungsfunktionen 72 bis 75 Speicherplatz beanspruchen, der beim Laden des Programmes noch nicht belegt war. Dadurch wächst der Programmspeicher dynamisch. Beachten Sie, daß das DX-Register aber nur den Speicherplatz verwaltet, den DOS beim Laden eines Programmes reserviert. Die durch die Funktionen 72 bis 75 zusätzlich gewonnene Kapazität bleibt unberücksichtigt. Daher ist die dynamische Speicherausweitung bei Programmen, die (zum Teil) im Speicher fest installiert werden sollen, mit Vorsicht anzuwenden.

17.2.4 Funktion 51 (hex 33): Crtl-Break-Abfrage testen oder festlegen

Funktion 51 meldet oder beeinflußt den Status der Ctrl-Break-Abfrage. Sie erinnern sich vielleicht, daß DOS eine Unterbrechung mit Ctrl-Break nur unter bestimmten Umständen bearbeitet (siehe Kapitel 15.2.2). Ab Version 2.00 läßt sich DOS so einstellen, daß die Unterbrechung zu jedem Zeitpunkt erkannt und bearbeitet wird. Aus Kompatibilitätsgründen gegenüber DOS 1 muß der erweiterte Ctrl-Break-Test optional sein. Die Festlegung, daß Ctrl-Break ständig abgefragt werden soll, kann auf drei Arten geschehen: mit dem Kommando BREAK, über die CONFIG.SYS-Datei oder mit DOS-Funktion 51.
Beim Aufruf der Funktion wird in Register AL festgelegt, ob der Status der Unterbrechungsabfrage festgestellt (AL gleich 0) oder verändert (AL gleich 1) werden soll. Die Statuskodierung in Register DL ist in beiden Fällen die gleiche. Der Wert 1 in DL bedeutet, daß die erweiterte Ctrl-Break-Abfrage aktiv ist bzw. aktiviert wird, der Wert 0 zeigt an, daß die Abfrage unterdrückt wird.

17.2.5 Funktion 53 (hex 35): Interruptvektor feststellen

Nach Aufruf von Funktion 53 steht im Registerpaar ES:BX der Interruptvektor, der zu der vor dem Aufruf in AL spezifizierten Interruptnummer gehört. Somit läßt sich überprüfen, zu welcher Routine bei Aufruf des betreffenden Interrupts verzweigt wird.
Für die Funktion gibt es eine vielfältige Anwendungspalette. Vermutlich am häufigsten wird sie verwendet, um einen Interruptvektor zu holen und abzuspeichern, bevor er mit Funktion 37 (hex 25) verändert wird. Zu einem späteren Zeitpunkt kann der Vektor wieder auf den ursprünglichen Wert zurückgesetzt werden. Eine andere Anwendung ist der Zugriff auf Tabellen über Vektoren. Um Felder in der Tabelle anzusprechen, ist es notwendig, den Wert des Tabellen- oder des Feldvektors zu kennen. Schließlich benötigt man manchmal die Information, auf welche Adresse ein Interruptvektor zeigt, das heißt, welche Routine dem Vektor zugeordnet ist (z.B. Standard- oder eigene Routine).

17.2.6 Funktion 54 (hex 36): Freie Diskettenkapazität feststellen

Funktion 54 stellt eine Reihe interessanter Informationen über ein Laufwerk zur Verfügung. Die Routine geht über das hinaus, was der Routinenname vermuten läßt. Das Laufwerk wird in Register DL mit einer Numerierung festgelegt, die von der üblichen abweicht: 0 = Standardlaufwerk, 1 = Laufwerk A, 2 = Laufwerk B usw. Wie bei allen DOS-Routinen (im Gegensatz zu den BIOS-Routinen) bezieht sich die Zuordnung auf logische, nicht auf physikalische Laufwerke.
Ein Fehler (z.B. ungültiges Laufwerk) wird durch den Wert hex FFFF in Register AX angezeigt. Nach einem korrekten Aufruf finden wir in Register AX die Sektoren pro Belegungseinheit (Cluster), in CX die Bytes pro Sektor, in BX die Anzahl der verfügbaren Cluster und DX enthält die Gesamtzahl der Cluster.
Aus diesen Angaben lassen sich weitere Daten über die Diskette errechnen:

```
CX * AX              'Bytes pro Belegungseinheit (Cluster)
CX * AX * BX         'freie Speicherkapazität (in Byte)
CX * AX * DX         'Gesamtspeicherkapazität
(BX * 100) / DX      'freier Speicher in Prozent der Gesamtkapazität
```

Wenn L die Länge einer Datei in Byte ist, kann die Anzahl der benötigten Sektoren mit folgender Formel errechnet werden:

```
(L + CX * AX -1) / (CX * AX)
```

Mit ähnlichen Formeln läßt sich feststellen, wieviel Belegungseinheiten und wieviel nicht benötigten Speicherplatz eine Datei belegt.

17.2.7 Funktion 56 (hex 38): Landesabhängige Information lesen

Funktion 56 ermöglicht es Programmen, unterschiedliche nationale Währungen und Zeitformate zu berücksichtigen. DOS stellt eine Tabelle mit landesspezifischen Daten bereit. DOS Version 2 liest nur einen kleinen Teil der Information aus der Tabelle, während DOS Version 3 alle relevanten Daten zur Verfügung stellt.
Register AL muß den Wert 0 enthalten, um die Standardlandesinformation zu erhalten. Ab Version 3.00 kann das Register auf einen vordefinierten Ländercode gesetzt werden. Um auf mehr als 255 Ländercodes zugreifen zu können, wird Register AL auf den Wert 255 (hex FF) gesetzt und der Ländercode in Register BX gegeben. Ist der Ländercode ungültig, setzt DOS die Carry-Flagge (CF) und der Fehlercode erscheint in Register AX. Andernfalls enthält BX den Ländercode und ein 32-byte-Bereich, auf den das Registerpaar DS:DX verweist, ist mit landesspezifischen Informationen gefüllt.
Programme, die unter DOS 2 ablaufen, können nur die Standardlandesdaten abfragen, indem AL auf Null gesetzt wird. Unter DOS 3 können die Information jedes Landes, dessen Code (AL = Code) in der Tabelle steht, gelesen werden. Ein Programm kann nach gültigen Codes suchen, indem es alle Codes testet und die brauchbaren zurückmeldet.

Feld	Offset	Länge (Byte)	Inhalt
1	0	2	Datums- und Zeitcode
2	2	2	Währungssymbol-String (ASCIIZ-Format)
3	4	2	Tausendertrennzeichen-String (ASCIIZ-Format)
4	6	2	Dezimalzeichen-String (ASCIIZ-Format)
5	8	24	Unbenutzt

Tabelle 17-5 Die landesabhängigen Informationen in DOS 2; die Daten sind in einem 32-byte-Bereich untergebracht, auf den der Zeiger DS:DX deutet

Feld	Offsett	Länge (Byte)	Inhalt
1	0	2	Datums- und Zeitcode
2	2	5	Währungssymbol-String (ASCIIZ-Format)
3	7	2	Tausendertrennzeichen-String (ASCIIZ-Format)
4	9	2	Dezimalzeichen-String (ASCIIZ-Format)
5	11	2	Datumstrennzeichen-String (ASCIIZ-Format)
6	13	2	Zeittrennzeichen-String (ASCIIZ-Format)
7	15	1	Plazierung des Währungssymbols: 1 = vorne, 0 = hinten
8	16	1	Nachkommastellen bei Währungsbeträgen
9	17	1	Zeitformat: 0 = 12-Stunden-Uhr; 1 = 24-Stunden-Uhr
10	18	4	Groß-/Kleinschreibungsfestlegung
11	22	2	Auflistungstrennzeichen (ASCIIZ-Format)
12	24	8	Reserviert

Tabelle 17-6 Die landesabhängigen Informationen ab DOS 3, wie sie durch Funktion 56 (hex 38) gemeldet werden (AL enthält den Ländercode)

Der 32-byte-Bereich mit landesspezifischen Informationen ist in DOS 2 anders kodiert als in DOS 3. Obwohl die Codes gleich zu sein scheinen, sind die beiden Formate nicht kompatibel.
Feld 1 (beide Formate) enthält ein Ganzzahlwort, das das Anzeigeformat von Datum und Zeit spezifiziert. Drei festgelegte Werte bestimmen die Formate von Datums- und Zeitangaben. Die Kodierung ist so ausgelegt, daß zukünftig weitere Formate hinzugefügt werden können.
In DOS 2 enthalten die nächsten drei Felder drei ASCIIZ-Strings, die aus je zwei Bytes bestehen: einem Daten-Byte, das das Format festlegt, und einem Null-Byte als Abschluß des ASCIIZ-Strings.

Code	Verwendung	Zeit	Datum
0	Amerika	H:M:S	M-T-J
1	Europa	H:M:S	T.M.J
2	Japan	H:M:S	J-M-T

Es bedeuten:
H = Stunde
M = Minute (für Zeit) / Monat (für Datum)
S = Sekunde
T = Tag
J = Jahr

Tabelle 17-7 Die drei festgelegten Datums- und Zeitformate, deren Codes in den ersten zwei Bytes des 32-byte-Bereiches, auf den DS:DX verweist, stehen

Feld 2 enthält das Währungssymbol des betreffenden Landes. Das Symbol darf in DOS 2 nur aus einem einzigen Zeichen bestehen (z.B. "$" für Dollar), was deutschen Verhältnissen nicht gerecht wird ("DM" benötigt zwei Bytes).
Feld 3 beinhaltet das Tausendertrennzeichen. Das ist das Zeichen, mit dem in großen Zahlen jeweils drei Stellen abgetrennt werden. In Deutschland verwendet man den Punkt (z.B. wird eine Million wie folgt dargestellt: 1.000.000).
Feld 4 enthält das Dezimalzeichen des betreffenden Landes, in Deutschland ein Komma (z.B. 3,5 für drei komma fünf).
Die restlichen 24 Bytes sind in DOS 2 ohne Bedeutung.
Das DOS-3-Format ist in Feld 1 (Datum und Zeit) mit DOS 2 identisch. Es folgen die drei ASCIIZ-Zeichenketten: Währung, Tausendertrennung und Dezimalzeichen. Im Unterschied zu DOS 2 stehen im Währungsfeld fünf Bytes zur Verfügung. Da das letzte für das Null-Zeichen benötigt wird, darf die Währungsabkürzung bis zu vier Zeichen lang sein. Das ermöglicht die Angabe von "DM" für Deutsche Mark.
Die Felder 5 und 6 weisen eine Länge von je zwei Bytes auf. Das erste Feld legt das Trennzeichen bei Datumsangaben (z.B. "." in 4.12.86), das zweite das Trennzeichen bei Zeitangaben (z.B. ":" in 11:34) fest.

Feld 7 ist ein einzelnes Byte, das spezifiziert, ob das Währungssymbol bzw. die Währungsabkürzung vor den Betrag (1) oder dahinter (0) geschrieben werden soll.
Feld 8 enthält eine 1-byte-Ganzzahl mit der Anzahl der in der Währung relevanten Nachkommastellen. DM-Beträge werden grundsätzlich mit zwei Dezimalstellen berechnet (Wert 2 in Feld 8); das gilt auch, wenn der Pfennigteil weggelassen wird. Im Unterschied dazu wird die italienische Lire ohne Nachkommastellen berechnet (Wert 0).
Feld 9 ist ein 1-byte-Feld zur Festlegung der Uhrzeitkonventionen. Bislang wird nur das erste Bit (Bit 0) verwendet. Ist es gesetzt (1), geht DOS von einer 24-Stunden-Uhr aus, andernfalls (0) von einer 12-Stunden-Uhr. In zukünftigen DOS-Versionen dürften vermutlich weitere Kodierungen hinzukommen.
Feld 10 enthält eine segmentierte Adresse von vier Bytes, die auf eine Routine zeigt, die die Verwendung von Groß- und Kleinbuchstaben festlegt.
Feld 11 enthält einen 2-byte-ASCIIZ-String, der das Zeichen enthält, das als Trennzeichen in Aufzählungen dient. Im Deutschen ist es das Komma (z.B. A, B, C, D und E).
Feld 12 umfaßt die verbleibenden acht Bytes des 32-byte-Bereiches und ist für zukünftige Zwecke (neue DOS-Versionen) reserviert.

17.2.8 Funktion 57 (hex 39): Unterverzeichnis anlegen (MKDIR)

Funktion 57 legt wie das DOS-Kommando MKDIR ein Unterverzeichnis (*Subdirectory*) an. Das Registerpaar DS:DX muß auf eine ASCIIZ-Zeichenkette verweisen, die den Pfadnamen des neuen Unterverzeichnisses enthält. Fehler werden im Standardformat für erweiterte Funktionen (siehe Kapitel 17.1.2) in Register AX gemeldet. Die möglichen Fehlercodes sind 3 (*Pfad nicht gefunden*) und 5 (*Zugriff nicht möglich*).

17.2.9 Funktion 58 (hex 3A): Unterverzeichnis löschen (RMDIR)

Funktion 58 löscht wie das DOS-Kommando RMDIR ein Unterverzeichnis. Es muß ein Pfadname als ASCIIZ-String angegeben sein, auf den das Registerpaar DS:DX zeigt. Fehler werden in Register AX gemeldet, wobei der Standardcode für erweiterte Befehle (siehe Kapitel 17.1.2) gilt. Die hier auftretenden Fehlercodes beschränken sich auf 3 (*Pfad nicht gefunden*) und 5 (*Zugriff nicht möglich*).
Es werden weder das aktuelle Verzeichnis noch Verzeichnisse, die Dateien oder Unterverzeichnisse enthalten, gelöscht. In diesen Fällen wird der Fehlercode 6 (*Versuch, aktuelles Verzeichnis zu löschen*) gemeldet.

17.2.10 Funktion 59 (hex 3B): Aktuelles Verzeichnis ändern (CHDIR)

Funktion 59 ändert analog dem DOS-Kommando CHDIR das aktuelle Verzeichnis. DS:DX muß auf einen Eingabestring im ASCIIZ-Format zeigen, der den Pfadnamen des neuen Verzeichnisses enthält. In AX steht der Standardfehlercode für erweiterte Funktionen (siehe Kapitel 17.12). Die einzige hier in Frage kommende Fehlermeldung ist die Nummer 3 (*Pfad nicht gefunden*).

17.2.11 Funktion 60 (hex 3C): Datei anlegen (CREAT)

Funktion 60 sucht nach einer Datei des vorgegebenen Namens und öffnet sie. Wird keine Datei gefunden, legt die Funktion eine neue Datei mit dem spezifizierten Namen an. Das ist die Standardoperation zum Öffnen von Ausgabedateien. Funktion 60 entspricht der traditionellen DOS-Funktion 22 (siehe Kapitel 16.1.23).
Das Registerpaar DS:DX muß auf einen ASCIIZ-String zeigen, der den Pfadnamen und den Dateinamen enthält. CX, oder eigentlich das Halbregister CL, enthält die Dateiattribute (nähere Erläuterungen zu Dateiattributen finden Sie in Kapitel 5.5.2.3). Die Dateinummer wird in Register AX bestimmt.
Obgleich sich mit Funktion 60 jede Datei sowohl für schreibenden als auch für lesenden Zugriff öffnen bzw. anlegen läßt, ist die Routine nur für Ausgabedateien geeignet. Die Länge einer bereits vorhandenen Datei wird auf Null gesetzt.
Die möglichen Fehlercodes sind: 3 (*Pfad nicht gefunden*), 4 (*keine Dateinummer verfügbar*) oder 5 (*Zugriff nicht möglich*). Der Code 5 zeigt, daß entweder kein Platz mehr für einen neuen Verzeichniseintrag vorhanden oder aber die existierende Datei als Nur-Lese-Datei gekennzeichnet ist und nicht für eine Ausgabe geöffnet werden kann. Da das Register AX schon für die Rückgabe der Dateinummer verwendet wird, müssen und können wir uns in diesem Fall für die Fehleranzeige auf die Carry-Flagge verlassen (in Kapitel 17.1.2 finden Sie hierzu eine Erklärung).

17.2.12 Funktion 61 (hex 3D): Datei öffnen

Der Aufruf von Funktion 61 ist der generelle Weg zum Öffnen einer Datei. Pfadname und Dateiname werden als ASCIIZ-String bereitgestellt, auf den DS:DX zeigt. Durch einen Moduscode in Register AL wird festgelegt, wozu die Datei verwendet werden soll. Die acht Bits des Registers AL werden in vier Felder unterteilt.

Bit 7 6 5 4 3 2 1 0	Bedeutung
I	Vererbungsflagge (Inheritance; DOS 3)
. S S S	Sharing-Modus (DOS 3)
. . . . R . . .	Reserviert für zukünftige Zwecke (DOS 3)
. . . . . A A A	Zugriffscode (Access; DOS 2 und DOS 3)

Tabelle 17-8 Die Kodierung des Dateimodus beim Öffnen einer Datei mit Funktion 61 in Register AL

Der Öffnungsmodus von DOS 2 ist recht einfach. Es finden nur die Zugriffs-Bits (AAA) Verwendung, alle anderen Bits werden auf Null gesetzt.

Bit 2 1 0	Verwendung
0 0 0	(Nur)-Lesezugriff
0 0 1	(Nur)-Schreibzugriff
0 1 0	Schreib- oder Lesezugriff

Tabelle 17-9 Der Zugriffscode des Registers AL für Funktion 61

DOS 3 verwendet die Vererbungs- und Sharing-Codes neben dem Zugriffscode. Die Kodierung ist in DOS 3 komplizierter, weil die Probleme des Datei-Sharing berücksichtigt werden müssen.
Bit 7, das *Vererbungs-Bit*, zeigt an, ob abgeleitete Prozesse (Programme, die sozusagen als "Derivat" eines anderen Programmes laufen und somit abhängige Unterprogramme darstellen) durch Vererbung Zugriff auf die Datei erhalten. Ist das Bit 7 gleich 0, können Unterprogramm darauf zugreifen, steht das Bit 7 auf 1, ist der Zugriff auf die Datei dem Hauptprogramm vorbehalten und die Unterprogramme erhalten nicht *automatisch* ein Zugriffsrecht. Unabhängig davon kann vom Hauptprogramm aus ein Unterprogramm zum Dateizugriff berechtigt werden.
Die Bits 4 bis 6, die *Sharing-Modus-Bits* (SSS) legen den Ablauf fest, wenn versucht wird, eine bereits geöffnete Datei zu öffnen. Es gibt fünf Sharing-Modi: *Kompatibilitätsmodus* (SSS=000), *ablehnender Schreib-/Lesemodus* (SSS=001), *ablehnender Schreibmodus* (SSS=010), *ablehnender Lesemodus* (SSS=011) und *nicht-ablehnender Modus* (SSS=100). Der Sharing-Modus ist sehr komplex und soll hier nicht weiter behandelt werden.
Bit 3 ist für zukünftige DOS-Versionen reserviert und sollte auf Null stehen.
Beachten Sie, daß weder ein Dateiattribut noch die Datensatzlänge spezifiziert werden. Funktion 61 findet jede existierende Datei, auch wenn sie geschützt ist und legt die Datensatzlänge standardmäßig auf ein Byte fest.

Die Fehlercodes sind: 2 (*Datei nicht gefunden*), 4 (*keine Dateinummer verfügbar*), 5 (*Zugriff nicht möglich*) und 12 (*ungültiger Zugriffscode*). Da Register AX sowohl für die Dateinummer als auch für den Fehlercode verwendet wird, muß man sich bei Fehlern auf die Carry-Flagge verlassen (siehe Kapitel 17.1.2).

17.2.13 Funktion 62 (hex 3E): Datei schließen

Funktion 62 schließt eine Datei und löscht alle Dateipuffer, die mit der in BX spezifizierten Dateinummer verbunden sind. BX sollte die Dateinummer enthalten, die beim letzten Öffnen der Datei gemeldet wurde. Der einzig mögliche Fehlercode ist 6 (*ungültige Dateinummer*).

17.2.14 Funktion 63 (hex 3F): Lesen (Datei oder Gerät)

Funktion 63 liest Daten aus der Datei, die mit der Dateinummer in BX korrespondiert. Der Lesevorgang kann sich auf ein beliebiges geeignetes Gerät beziehen, das wie eine Datei behandelt wird. Die Anzahl der zu lesenden Bytes wird in CX festgelegt, das Registerpaar DS:DX muß auf den Puffer im RAM-Speicher zeigen, in den die Daten eingelesen werden sollen. Nach Aufruf der Funktion enthält AX die Anzahl der gelesenen Bytes, ist der Wert Null, wurde versucht, vom Ende einer Datei zu lesen.
Die möglichen Fehlercodes sind: 5 (*Zugriff nicht möglich*) und 6 (*ungültige Dateinummer*). Auch hier müssen und können wir uns zur Fehlererkennung auf die Carry-Flagge verlassen, da das Register AX sowohl für den Fehlercode als auch für die Anzahl der gelesenen Bytes verwendet wird. Sehen Sie hierzu auch in Kapitel 17.1.2 nach.

17.2.15 Funktion 64 (hex 40): Schreiben (Datei oder Gerät)

Funktion 64 schreibt Daten in die Datei, deren Dateinummer im Register BX spezifiziert ist. Die Ausgabe kann auf ein beliebiges geeignetes Gerät umgelenkt werden, das wie eine Datei behandelt wird. CX gibt die Anzahl der zu schreibenden Bytes an, das Registerpaar DS:DX zeigt auf die Anfangsadresse der Datenbytes im RAM-Speicher. Nachdem die Funktion ausgeführt wurde, enthält AX die Anzahl der geschriebenen Bytes.
Ein Fehler wird wie üblich über die Carry-Flagge gemeldet. Register AX enthält als Fehlercode entweder 5 (*Zugriff nicht möglich*) oder 6 (*ungültige Dateinummer*). Auch wenn die Carry-Flagge nicht auf einen Fehler hinweist, kann dennoch einer aufgetreten sein. Die Flagge meldet nämlich nicht, wenn auf dem Datenträger nicht mehr genügend Speicherplatz vorhanden ist, um die gesamte Datei unterzubringen. Um zu überprüfen, ob dieser Fehler auftrat, muß die Anzahl der geschriebenen Bytes (in AX)

mit der Anzahl der zu schreibenden Bytes (in CX) verglichen werden. Nur wenn beide Werte übereinstimmen, ist die Schreiboperation erfolgreich verlaufen.

17.2.16 Funktion 65 (hex 41): Datei löschen

Funktion 65 löscht den Verzeichniseintrag einer Datei, so daß der Dateiname nicht mehr im Verzeichnis erscheint. Die Datei wird durch eine ASCIIZ-Zeichenkette, die Pfadnamen und Dateinamen enthält, spezifiziert. Das Registerpaar DS:DX zeigt auf den String.
Nur-Lesedateien können mit Funktion 65 nicht gelöscht werden. Um eine solche Datei zu löschen, muß zuerst mit Funktion 67 (hex 43) das Dateiattribut auf Null geändert werden. Die Dateigruppenzeichen "?" und "*" dürfen nicht verwendet werden.
Die möglichen Fehlercodes sind 2 (*Datei nicht gefunden*) und 5 (*Zugriff nicht möglich*).

17.2.17 Funktion 66 (hex 42): Dateizeiger bewegen

Funktion 66 wird verwendet, um die logische Schreib-/Leseposition in einer Datei zu ändern. Die Dateinummer muß in BX stehen, die Ausgangsposition des Dateizeigers in AL und die Anzahl der Bytes, um die der Zeiger bewegt werden soll, in CX:DX. Der Byte-Offset in CX:DX ist eine vorzeichenlose Ganzzahl mit 32 Bits, wobei CX den höherwertigen Teil (der Null ist, es sei denn, der Offset geht über 65.535 hinaus) und DX den niederwertigen Teil enthält.
Für die Festlegung der Ausgangsposition in Register AL gibt es drei Alternativen. Ist AL gleich 0, wird der Offset vom Dateianfang aus gerechnet. Bei AL gleich 1 wird der Offset auf die aktuelle Zeigerposition bezogen. Steht in AL der Wert 2, wird der Offset vom Dateiende aus gerechnet. Wenn man in AL eine 2 ablegt und den Offset in CX:DX auf 0 setzt, läßt sich die Länge der Datei feststellen. Mit AL gleich 0 und CX:DX ebenfalls 0 wird der Dateizeiger an den Anfang der Datei gesetzt.
Nach Ausführung der Funktion enthält das Registerpaar DX:AX die momentane Zeigerposition in Bezug auf den Beginn der Datei. Der Zeiger wird als 32-bit-Ganzzahl übergeben, der höherwertige Teil befindet sich in Register DX, der niederwertige Teil in AX. Beachten Sie die ungewöhnliche Wahl des Registerpaares.
Mögliche Fehlercodes sind: 1 (*ungültige Angabe der Ausgangsposition*) und 6 (*ungültige Dateinummer*). Auch hier müssen und können wir uns bezüglich einer Fehlermeldung auf die Carry-Flagge verlassen, da das AX-Register neben dem Fehlercode auch einen Teil der neuen Zeigerposition beinhaltet. Eine Erklärung zur Benutzung der Carry-Flagge bei Fehlermeldungen finden Sie in Kapitel 17.1.2.

17.2.18 Funktion 67 (hex 43): Dateiattribute festlegen/feststellen (CHMOD)

Funktion 67 setzt oder meldet die Attribute einer Datei (in Kapitel 5.5.2.3 finden Sie nähere Informationen zu den Dateiattributen). DS:DX zeigt auf eine ASCIIZ-Zeichenkette, die den Dateinamen enthält. Die Dateigruppenzeichen "?" und "*" sind nicht erlaubt. Enthält Register AL beim Funktionsaufruf den Wert 0, werden die Dateiattribute gelesen und in CX abgelegt. Bei AL gleich 1 versieht die Funktion die Datei mit den in CX stehenden Attributen. In beiden Fällen wird der Attributcode nur aus dem C-Halbregister CL genommen.
Die möglichen Fehlercodes in AX sind 2 (*Datei nicht gefunden*), 3 (*Pfad nicht gefunden*) und 5 (*Zugriff nicht möglich*).

17.2.19 Funktion 68 (hex 44): Ein-/Ausgabesteuerung für Geräte (IOCTL)

Mit Funktion 68 lassen sich diverse Ein- und Ausgabeoperationen durchführen, von denen sich die meisten auf Geräte beziehen. Über AL wird eine der zehn Routinen angewählt, die die Ziffern 0 bis 8 und 11 tragen (8 und 11 sind nur für DOS 3 bestimmt). BX meldet die Dateinummer.

Code	Beschreibung
0	Geräteinformationen feststellen (in DX)
1	Geräteinformationen festlegen (aus DX, DH muß 0 sein)
2	Lesen (siehe Text)
3	Schreiben (siehe Text)
4	Von Laufwerk lesen (siehe Text)
5	Auf Laufwerk schreiben (siehe Text)
6	Eingabestatus feststellen (siehe Text)
7	Ausgabestatus feststellen (siehe Text)
8	Test, ob Datenträger austauschbar; nur DOS 3 (siehe Text)
11	Sharing-Eintrag ändern; nur DOS 3 (siehe Text)

Tabelle 17-10 Die zehn Unterfunktionen, die von Funktion 68 bereitgestellt werden. Auswahl erfolgt über das Register AL

Unterfunktionen 0 und 1: Diese Funktionen setzen und melden die Geräteinformationen, die in DX als komplexe Bit-Kodierung enthalten sind. Bit 7 steht für Geräte auf 1, für Dateien auf 0. Wie die Bits 0 bis 5 für Geräte kodiert sind, entnehmen Sie bitte der Tabelle 17-11. Bei Dateien stellen diese Bits die Laufwerksnummer dar: 0 bezeichnet Laufwerk A, 1 Laufwerk B usw.

Bit F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Bedeutung
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	1 = Standardkonsoleneingabe
.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	1 = Standardkonsolenausgabe
.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	1 = Nullgerät (benötigt keine Ein-/Ausgabe)
.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	1 = Uhr
.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	1 = Spezialgerät (was immer das auch sein mag)
.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	Datenübergabe: 1 = unverarbeitet; 0 = verarbeitet
.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	0 = Ende der Datei; 1 = kein Dateiende (für Eingabe)
.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	1 = Gerät; 0 = Laufwerksdaei
.	.	.	.	.	.	.	R	.	.	.	.	.	.	.	.	Reserviert
.	.	.	.	.	.	R	.	.	.	.	.	.	.	.	.	Reserviert
.	.	.	.	.	R	.	.	.	.	.	.	.	.	.	.	Reserviert
.	.	.	.	R	.	.	.	.	.	.	.	.	.	.	.	Reserviert
.	.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	Reserviert
.	.	R	.	.	.	.	.	.	.	.	.	.	.	.	.	Reserviert
.	x	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1 = Gerät kann Kontrollzeichenketten während des Schreibens und Lesens verarbeiten
R	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	Reserviert

Tabelle 17-11 Die Bits des Registers DX für die Unterfunktionen 0 und 1 der Funktion 68

Unterfunktionen 2 bis 5: In allen vier Funktionen wird in CX die Anzahl der Bytes, die zum Steuerkanal des Laufwerkes geschrieben oder aus ihm gelesen werden sollen, spezifiziert. Das bezieht sich nicht auf Dateizugriffe, sondern nur auf den Kanal. Das Registerpaar DS:DX zeigt auf den Datenbereich, in dem gelesen bzw. aus dem geschrieben werden soll. Für die Unterfunktionen 4 und 5 enthält BL die Laufwerksnummer (0 = Standard, 1 = Laufwerk A, 2 = Laufwerk B, usw.).
Mit den Routinen können nur dann Daten geschrieben oder gelesen werden, wenn das Gerät oder das Laufwerk Kontrollzeichenketten verarbeiten kann. Der Schreib-/Lesestatus wird durch Bit 14 des Registers DX angezeigt.
Unterfunktionen 6 und 7: Diese Unterfunktionen melden den Eingabe- (Routine 6) bzw. Ausgabestatus (Routine 7) in Register AX. Bei AX gleich 255 (hex FF) ist das Gerät/Laufwerk zur Ein- bzw. Ausgabe bereit, bei AX gleich 0 ist es nicht bereit.
Unterfunktion 8: Unterfunktion 8, die erst ab DOS 3.00 verfügbar ist, zeigt an, ob das Gerät mit austauschbaren Datenträgern arbeitet oder nicht. Beispiel: Eine Diskettenstation arbeitet mit einem austauschbaren Datenträger (man kann eine Diskette entnehmen und eine andere einlegen), bei einer Festplattenstation liegt ein Datenträger vor, der nicht austauschbar ist. Diese Unterfunktion ist sehr hilfreich, wenn ein Programm die Ein- und Ausgabe lückenlos überwachen soll. Ein Gerät mit auswechselbarem Datenträger wird durch den Wert 0 in Register AX

angezeigt, ein anderes durch den Wert 1. Steht in AX der Wert 15 (hex F), wurde ein ungültiger Gerätecode angegeben.
Unterfunktion 11 (hex B): Die Funktion ist erst ab DOS 3.00 vorhanden und erlaubt die Kontrolle über Datei-Sharing-Probleme. Man kann sich leicht vorstellen, daß es zu Schwierigkeiten führen kann, wenn mehrere Benutzer zur gleichen Zeit auf eine Datei zugreifen wollen. Wenn jeder Benutzer die Datei nur kurzzeitig beansprucht, lösen sich die Problem im Zeitverlauf von selbst. Voraussetzung dafür ist, daß DOS mehrmals versucht, auf eine Datei zuzugreifen, bevor ein Fehler (*Zugriff nicht möglich*) gemeldet wird. Mit Unterfunktion 11 kann man festlegen, wieviele Zugriffsversuche DOS vornehmen soll, bevor eine Fehlermeldung ausgegeben wird. Die Anzahl der Versuche wird in DX und das Zeitintervall zwischen den Versuchen durch einen Zählerstand in CX festgelegt.

17.2.20 Funktion 69 (hex 45): Dateinummern duplizieren (DUP)

Funktion 69 dupliziert die Dateinummer einer offenen Datei und ordnet der gleichen Datei eine neue zweite Dateinummer zu. Statt um eine Datei kann es sich um ein beliebiges geeignetes Gerät handeln. Alle Operationen, die mit der *einen* Dateinummer durchgeführt werden, wirken sich in gleicher Weise auf die *zweite* Dateinummer aus (siehe auch Funktion 70).
Das BX-Register enthält die alte und AX die neue Dateinummer. Die möglichen Fehlercodes sind 4 (*keine Dateinummer verfügbar*) und 6 (*ungültige Dateinummer*).
Bis heute ist dem Autor keine sinnvolle Verwendung für die Funktion bekannt, sicherlich wird es aber eine geben. Oder sollte etwa ...?

17.2.21 Funktion 70 (hex 46): CDUP - Dateinummernduplizierung erzwingen

Funktion 70 stellt ähnlich wie Funktion 69 ein Duplikat einer Dateinummer zur Verfügung. Es muß aber eine zweite bereits existierende Dateinummer bereitgestellt werden, da von DOS keine neue angelegt wird. Die zweite Dateinummer wird im allgemeinen schon einem anderen Zweck gedient haben. Bezieht sich die zweite Nummer auf eine geöffnete Datei, wird die Datei geschlossen, bevor die Funktion ausgeführt wird. Nach Aufruf der Funktion beziehen sich alle Operationen, die mit der einen Dateinummer durchgeführt werden, auch auf die zweite. Die beiden Dateinummern sind fest miteinander gekoppelt.
Das BX-Register enthält die erste Dateinummer, CX die zweite. Der einzig mögliche Fehlercode ist 6 (*ungültige Dateinummer*).
Auch diese Routine scheint genau wie Funktion 69 wenig sinnvoll einsetzbar zu sein. Der Schein trügt, es gibt einige Anwendungsmöglichkeiten. So kann Funktion 70 benutzt werden, um die Ein- oder Ausgabe

auf eines der Standard-E/A-Geräte umzulenken. Angenommen, in einem Programm soll die Ausgabe auf Drucker zeitweise in eine Datei umgeleitet werden. Dazu wird die Datei geöffnet, die von DOS zugeordnete Dateinummer ist aus BX zu ersehen. Diese Dateinummer wird auf die Standarddrucker-Dateinummer dupliziert, indem die Dateinummer 4 in CX geladen wird. Von nun an geht die auf den Standarddrucker gerichtete Ausgabe in Wirklichkeit in die Datei. Um den Ursprungszustand wieder herzustellen, aber die Datei dennoch für spätere Zwecke in Bereitschaft zu halten, wird die Druckerdateinummer auf eine dritte Dateinummer dupliziert, bevor die neue Dateinummer auf die Druckerdateinummer kopiert wird. Unter Funktion 75 ist eine Anwendungsmöglichkeit für diese Technik beschrieben. Sie sollten sie jedoch nur benutzen, wenn Sie sie tatsächlich verstanden haben.

17.2.22 Funktion 71 (hex 47): Aktuelles Verzeichnis melden

Funktion 71 meldet das aktuelle Verzeichnis in Form eines ASCIIZ-Strings. Die Laufwerksnummer wird in Register DL spezifiziert (0 = Standardlaufwerk, 1 = Laufwerk A, 2 = Laufwerk B). Das Registerpaar DS:SI zeigt auf den Datenbereich, der den Pfadnamen aufnehmen soll und bis zu 64 Bytes lang sein kann. DOS meldet den Pfadnamen des aktuellen Verzeichnisses, einschließlich des Stamm- oder Urverzeichnisses. Auf den Pfadnamen folgt ein Null-Byte als Abschluß des ASCIIZ-Strings.
Der Ergebnisstring enthält zwar den vollständigen Pfadnamen, aber nicht den Laufwerksbuchstaben mit Doppelpunkt und nicht den ersten Schrägstrich (wie in *A:*). Falls es sich bei dem aktuellen Verzeichnis um ein Stammverzeichnis handelt, ist der Ergebnisstring daher ein Nullstring, der kein Zeichen enthält. In Programmen ist es sinnvoll, die Zeichenkette aufzubereiten, das heißt, Laufwerksbuchstaben, Doppelpunkt und Schrägstrich hinzuzufügen.
Der einzig mögliche Fehlercode ist 15 (*ungültige Laufwerksspezifikation*).

17.2.23 Funktion 72 (hex 48): Speicherbereich belegen

Funktion 72 dient der dynamischen Speicherbelegung. Mit der Funktion wird ein Teil des Hauptspeichers der DOS-Kontrolle entzogen. In Register BX wird *die Anzahl* der benötigten Speichersegmente (in 16-byte-Einheiten) festgelegt. Nach Aufruf der Funktion enthält AX die Segmentadresse des belegten Speicherblocks.
Die möglichen Fehlercodes sind 7 (*Speicherkontrollblöcke zerstört*) und 8 (*zu wenig Speicherplatz*). Verläuft die Belegung erfolgreich, steht in Register BX die Größe des längsten zusammenhängenden freien Speicherblocks.

17.2.24 Funktion 73 (hex 49): Speicherbereich freigeben

Mit Funktion 73 läßt sich ein Speicherbereich, der mit Funktion 72 belegt wurde, wieder an DOS zurückgeben. Das ES-Register muß auf die Segmentadresse des Blocks, der freigegeben werden soll, zeigen. Das ist derselbe Wert, der von Funktion 72 in Register AX geladen wird.
Die möglichen Fehlercodes sind 7 (*Speicherkontrollblocks zerstört*) und 9 (*ungültige Speicherblockadresse*).

17.2.25 Funktion 74 (hex 4A): Größe des belegten Speicherbereichs verändern (SETBLOCK)

Mit Funktion 74 läßt sich ein Speicherbereich, der mit Funktion 72 belegt wurde, vergrößern oder verkleinern. Das ES-Register zeigt auf die Segmentadresse des Bereiches, der verändert werden soll. Das Register BX enthält die neue Speichergröße in Segmenteinheiten (Einheiten von 16 Bytes).
Ist nicht mehr genügend Platz für eine Vergrößerung vorhanden, zeigt BX die Länge des größten verfügbaren Speicherbereichs (in Segmenten) an.
Als Fehlercodes kommen in Frage: 7 (*Speicherkontrollblöcke zerstört*), 8 (*zu wenig Speicherplatz*) und 9 (*ungültige Speicherblockadresse*).

17.2.26 Funktion 75 (hex 4B): Programm laden und ausführen (EXEC)

Die EXEC-Funktion erlaubt es, von einem Programm aus ein Unterprogramm in den Speicher zu laden und ausführen zu lassen. Das Registerpaar DS:DX muß auf einen ASCIIZ-String zeigen, der den Pfadnamen und den Dateinamen der zu ladenden Datei enthält. Das Registerpaar ES:BX verweist auf einen Parameterblock, der alle nötigen Informationen für die Ladeoperation enthält. In AL wird bestimmt, ob das Unterprogramm nur geladen oder auch automatisch gestartet werden soll.
Ist AL gleich 0, wird das Unterprogramm geladen und, nachdem ein Programmsegmentpräfix (PSP) angelegt ist, ausgeführt. Bei AL gleich 1 wird das Unterprogramm nur geladen, ohne daß ein PSP angelegt oder das Programm gestartet wird. Das Programm kann aber jederzeit abgearbeitet werden. Sehr große Programme mit mehreren Programmteilen werden auf diese Weise oftmals im Overlay-Verfahren abgearbeitet. *Overlay-Verfahren* bedeutet, daß immer nur ein Teil des Gesamtprogrammes im Hauptspeicher gehalten wird, die jeweils benötigten Teile werden einzeln nachgeladen und ausgeführt. Statt eines Programms lassen sich mit Routine 75 auch Daten in den Speicher laden.

Steht in AL eine 0, zeigt ES:BX auf einen 14 Bytes langen Block, der die Informationen enthält, die Sie in Tabelle 17-12 aufgelistet finden. Steht dort der Wert 1, ist dieser Block nur vier Bytes lang und enthält die in Tabelle 17-13 aufgeführten Informationen.

Offset	Länge (Byte)	Beschreibung
0	2	Segmentadresse des Umgebungsstrings
2	4	Segmentierter Zeiger auf Befehlszeile
6	4	Segmentierter Zeiger auf ersten Standard-FCB
10	4	Segmentierter Zeiger auf zweiten Standard-FCB

Tabelle 17-12 Die Informationen des EXEC-Kontrollblocks, wenn AL = 0. Zeiger ist das Registerpaar ES:DX. Die Auflistung bezieht sich auf das PSP des geladenen Programmes.

Offset	Länge (Byte)	Beschreibung
0	2	Segmentadresse, die angibt, wohin das Programm geladen werden soll
2	2	Verschiebungsfaktor für relokatible Programme (nur für Programme im EXE-Format)

Tabelle 17-13 Die Information des EXEC-Kontrollblocks, wenn AL = 1; Zeiger ist ES:DX

Wird ein Programm geladen und ausgeführt, sind die aktuellen Dateinummern auch dem Unterprogramm zugänglich. Wie schon unter Funktion 70 (hex 46) erwähnt, kann in einem Programm die Ein- und Ausgabe über Dateinummern so umgeleitet werden, daß davon auch die Unterprogramme betroffen sind. Ein Beispiel: Eine Standardroutine zum Sortieren von Dateien wird aufgerufen und gibt ihre Meldungen z.B. in eine Datei, die vom Hauptprogramm gelesen werden kann.
Bevor Funktion 75 aufgerufen wird, sollte man prüfen, ob genügend Speicherplatz zur Aufnahme des zu ladenden Programmes vorhanden ist. Dazu kann Funktion 74 (hex 4A) verwendet werden. Da der Ladeprozeß zum Teil durch den Kommandointerpreter ausgeführt wird, muß sich der Interpreterkern im Speicher befinden oder gegebenenfalls nachgeladen werden.
Unter den Programmen, die mit Funktion 75 geladen werden können, ist auch der DOS-Kommandointerpreter. Über einen Kommandostring läßt sich eine Stapelverarbeitungsdatei (Batch-Datei) zur Ausführung bringen. Diese Datei kann vom Ursprungsprogramm aus dynamisch angelegt worden sein. In der Stapelverarbeitungsdatei werden die gewünschten Programme aufgerufen und schließlich mit EXIT die Abarbeitung des zweiten Kommandointerpreters gestoppt. Das Ursprungsprogramm kann nun weiter ausgeführt werden. Der Vorgang ist sehr komplex, aber er eröffnet weitreichende Perspektiven.

Achtung: Funktion 75 zerstört die Inhalte der meisten Register einschließlich der Stapelregister SS und SP. Alle wichtigen Register sollten vor Ausführung der Funktion gesichert und danach zurückgespeichert werden. Es ist klar, daß der Stapel zu diesem Zweck nicht eingesetzt werden kann, da er selbst nicht unverändert bleibt. Um das Problem zu lösen, kann man die Registerinhalte in den Programmcode einbauen. Nach Aufruf von Funktion 75 holen die ersten Befehle die Inhalte wieder aus dem Programm heraus und speichern sie in den entsprechenden Registern ab. Diese Befehle können Registerspeicherbefehle sein, deren Argumente vor dem Funktionsaufruf im Programmcode geändert wurden. Allerdings - die dynamische Veränderung von Programmcode ist mit sauberen Programmiertechniken unvereinbar. Sie sollten solche Tricks, wenn möglich vermeiden. Läßt es sich nicht umgehen, weisen Sie in der Programmdokumentation deutlich darauf hin, um nicht zukünftige Programmänderungen unmöglich zu machen.

Die möglichen Fehlercodes sind 1 (*ungültige Funktionsnummer*), 2 (*Datei nicht gefunden*), 5 (*Zugriff nicht möglich*), 8 (*zu wenig Speicherplatz*), 10 (*ungültige Umgebung*) und 11 (*ungültiges Format*).

17.2.27 Funktion 76 (hex 4C): Erweitertes Programm beenden

Funktion 76 schließt alle mit den Funktionen 61 (hex 3D) und 60 geöffneten Dateien, beendet das laufende Programm und übergibt einen Statuscode in Register AL. Falls das Programm als Unterprogramm aufgerufen wurde, läßt sich der Statuscode mit Funktion 77 feststellen. Wurde das Programm als DOS-Kommando aufgerufen, kann der Statuscode in einer Stapelverarbeitungsdatei (Batch-Datei) mit ERRORLEVEL abgetestet werden.

17.2.28 Funktion 77 (hex 4D): Statuscode des Unterprogramms melden

Funktion 77 meldet den Statuscode des Programmes, das als Unterprogramm aufgerufen und mit Funktion 76 gestoppt wurde. Es wird eine zweiteilige Information gegeben: In AL befindet sich der Statuscode, der vom Programm übermittelt wird und AH zeigt an, wie das Programm beendet wurde. Es sind vier Resultate möglich: AH gleich 0 signalisiert einen normalen Programmstopp. Steht in AH der Wert 1, wurde die Abarbeitung durch Ctrl-Break abgebrochen. AH gleich 2 steht für einen nicht behebbaren Gerätefehler, der zum Programmabbruch führte. Ist in AH der Wert 3 zu finden, wurde die Funktion 49 (hex 32) *Beenden und im Speicher verbleiben* aufgerufen.

17.2.29 Funktion 78 (hex 4E): Dateisuche beginnen (FIND FIRST)

Funktion 78 sucht ein Verzeichnis nach der ersten Datei durch, die den vorgegebenen Spezifikationen entspricht. Pfadname und Dateiname müssen in einem ASCIIZ-String stehen, auf den DS:DX verweist. Im Dateinamen dürfen die Dateigruppenzeichen "*" und "?" vorkommen. CX (eigentlich CL) enthält die Dateiattributspezifikationen für die Dateisuche. Wird eine passende Datei gefunden, legt DOS 43 Informations-Bytes über die Datei im Diskettentransferbereich (DTA) ab.
Der ASCIIZ-String am Ende der Informations-Bytes enthält den Dateinamen, wie er im Verzeichnis steht. Dateiname (im engeren Sinne) und Namenserweiterung sind durch einen Punkt getrennt, ist keine Erweiterung vorhanden, fällt auch der Punkt weg.
Die Routine ist der traditionellen DOS-Funktion 17 (hex 11) sehr ähnlich. Die Verwendung der Dateiattribute in Funktion 78 entspricht der in Funktion 17 (siehe Kapitel 16.1.18).
Die Attributsuche folgt einer besonderen Logik. Spezifizieren Sie die Attribute *versteckt*, System oder Verzeichnis, werden sowohl alle normalen Dateien als auch alle mit den Attributen behafteten gesucht. Spezifizieren Sie einen Datenträgerkennsatz, werden nur Verzeichniseinträge mit diesem Attribut in die Suche einbezogen. Die Attribute *Archiv* und *Nur-Lesen* bleiben in jedem Fall unberücksichtigt. DOS-Versionen vor 2.00 verarbeiten keine Verzeichnis-, Datenträger-, Archiv- oder Nur-Lesen-Attribute.
Die Fehlercodes sind wie üblich in Register AX zu finden: 2 (*Datei nicht gefunden*), 18 (*keine weiteren Dateien vorhanden*). CF zeigt in beiden Fällen keine Fehlermeldung an.

Offset	Länge (Byte)	Beschreibung
0	21	Dient DOS zur Fortsetzung der Suche nach der nächsten passenden Datei (siehe Funktion 79)
21	1	Attribut der gefundenen Datei
22	2	Zeitangabe der Datei (siehe Kapitel 5.5.2.5)
24	2	Datumsangabe der Datei (siehe Kapitel 5.5.2.6)
26	4	Dateilänge in Bytes
30	13	Dateiname und Namenserweiterung (ASCIIZ String)

Tabelle 17-14 Der Informationsbereich im DTA nach Aufruf der Funktion 78 (hex 4E)

17.2.30 Funktion 79 (hex 4F): Dateisuche fortsetzen

Funktion 79 führt die Suche, die mit Funktion 78 eingeleitet wurde, weiter. Sie ist von den Informationen, die am Anfang des DTA stehen, abhängig, dieser Bereich sollte daher nicht überschrieben werden.

Als Fehlercode kann hier 18 (*keine weiteren Dateien vorhanden*) auftreten. Auch hier wird CF nicht gesetzt.
Lesen Sie bitte auch unter Funktion 78 nach.

17.2.31 Funktion 84 (hex 54): Verifizierstatus feststellen

Funktion 84 zeigt an, ob Daten, die auf Diskette geschrieben werden, automatisch überprüft (verifiziert) werden oder nicht. AL gleich 0 bedeutet, daß keine Schreibverifikation stattfindet, 1 das Gegenteil. Die Umschaltung geschieht mit Funktion 46 (hex 2E) (siehe Kapitel 16.1.43).
Funktion 84 bringt eine ärgerliche Inkonsistenz in die DOS-Funktionen. Während die Operationen *Feststellen* und *Festlegen* oftmals in einer Funktion zusammengefaßt sind (z.B. Funktion 86), existieren andererseits manchmal unterschiedliche Funktionen zum Festlegen und zum Feststellen eines Status (z.B. Funktionen 84 und 46).

17.2.32 Funktion 86 (hex 56): Datei umbenennen

Genau wie mit dem DOS-Kommando RENAME läßt sich mit Funktion 86 der Name einer Datei verändern. Darüberhinaus kann mit Funktion 86 ein Dateieintrag von einem Verzeichnis in ein anderes verlegt werden. Die Datei selbst wird dabei nicht verändert, sondern nur der Eintrag. Daraus folgt, daß sich der alte und der neue Verzeichnispfad auf das gleiche Laufwerk beziehen müssen. Es ist bedauerlich, daß das DOS-Kommando RENAME diese Möglichkeit nicht zur Verfügung stellt.
Die Funktion benötigt zwei Vorgaben: den alten und den neuen Dateinamen. Die Namen dürfen Laufwerksspezifikationen und Pfadkomponenten beinhalten. Allerdings müssen sich beide Laufwerksangaben auf das gleiche Laufwerk beziehen. Dadurch ist gewährleistet, daß Verzeichniseintrag und dazugehörige Datei stets im selben Laufwerk stehen. Die Dateigruppenzeichen "?" und "*" dürfen nicht verwendet werden, da sich die Funktion immer nur auf eine einzelne Datei bezieht.
Wie Sie vielleicht schon erwartet haben, werden beide Dateinamen in Form eines ASCIIZ-String mit einem Null-Byte am Ende angegeben. Das Registerpaar DS:DX zeigt auf die Zeichenkette des alten, das Registerpaar ES:DI auf die Zeichenkette des neuen Namens.
Die möglichen Fehlercodes sind 3 (*Pfad nicht gefunden*), 5 (*Zugriff nicht möglich*) und 17 (*anderes Laufwerk/Gerät*).

17.2.33 Funktion 87 (hex 57): Datum und Zeit für Datei stellen/ablesen

Funktion 87 setzt und meldet die Zeit- und Datumsangaben der Datei. Erinnern Sie sich: Jede Datei wird mit der aktuellen Zeit und dem Datum

versehen, wenn sie angelegt oder verändert wird. AL spezifiziert, ob die Zeit abgelesen (AL = 0) oder verändert (AL = 1) werden soll.
Die Datei wird über die Dateinummer in BX angewählt. Das bedeutet, daß die Routine für Dateien brauchbar ist, die mit einer erweiterten DOS-Funktion geöffnet wurden. Beachten Sie bitte weiterhin, daß die neuen Datums- und Zeitangaben nur dann von der Datei angenommen werden, wenn die Datei geschlossen wird.
Datum und Zeit werden in den Registern CX und DX im gleichen Format wie in den Verzeichniseinträgen abgelegt, allerdings in anderer Reihenfolge. Die Zeitangabe muß in Register DX, die Datumsangabe in Register DX stehen. Der höherwertige Teil steht in CH bzw. DH, der niederwertige in CL bzw. DL.
Datum und Zeit können mit folgenden Formeln aufgebaut oder zerlegt werden:

```
CX = STUNDE * 2048 + MINUTE * 32 + SEKUNDE / 2
DX = (JAHR -1980) * 512 + MONAT * 32 + TAG
```

Fehlercodes: 1 (*ungültige Funktionsnummer* - bezieht sich auf die Unterfunktion in AL, nicht auf die Hauptfunktionsnummer) und 6 (*ungültige Dateinummer*).

17.3 DOS-3-Erweiterungen

Die bislang in diesem Kapitel behandelten Funktionen wurden mit DOS Version 2.00 eingeführt und stehen seitdem zur Verfügung. DOS 3.00 brachte weitere fünf Funktionen und einige Verbesserungen der DOS-2-Funktionen mit sich.

17.3.1 Funktion 89 (hex 59): Erweiterten Fehlercode melden

Funktion 89 kann aufgerufen werden, wenn ein Fehler vorliegt. Sie liefert detaillierte Angaben über einen Fehler, der unter einem der folgenden Umstände aufgetreten ist: in einer Routine zur Bearbeitung fataler (kritischer) Fehler, wenn ein DOS-Funktionsaufruf (mit Interrupt 33 (hex 21)) mit der Carry-Flagge (CF) einen Fehler gemeldet hat oder nachdem eine FCB-Dateioperation einen Fehlercode von 255 (hex FF) geliefert hat. Die Funktion arbeitet nicht mit DOS-Funktionen, die die Carry-Flagge bei einem Fehler nicht setzen.
Die Routine wird wie alle anderen auch durch die Funktionsnummer in Register AH angewählt. Darüberhinaus ist in BX ein DOS-Versionscode anzugeben, für DOS 3.00 ist er 0.
Vier verschiedene Informationen werden gegeben: AX enthält den erweiterten Fehlercode, BH die Fehlerart, BL den Fortführungscode (der angibt, was nach dem Fehler zu tun ist) und CH den Fehlerort.

Der Fehlercode, der in AX ablesbar ist, wird in drei Gruppen unterteilt: Die Codes 1 bis 18 stehen für Fehler bei Funktionsaufrufen (Funktionen unter Interrupt 33), die Codes 19 bis 33 signalisieren fatale (kritische) Fehler (durch Interrupt 36) und die Codes 32 bis 83 zeigen Fehler aus DOS-3-Funktionen an. Der Code 0 bedeutet, daß die betreffende Routine keine Fehlercodes bereitstellt.

Code	Bedeutung	Code	Bedeutung
1	ungültige Funktionsnummer	20	unbekannte Einheitenangaben
2	Datei nicht gefunden	21	Diskettenlaufwerk nicht bereit
3	Pfad nicht gefunden	22	ungültiges Kommando
4	keine Dateinummer verfügbar	23	Laufwerksdatenfehler
5	Zugriff nicht möglich (z. B. Versuch, in eine Nur-Lesen-Datei zu schreiben)	24	falsche Strukturlänge
		25	Laufwerkssuchfehler
6	ungültige Dateinummer	26	unbekannter Datenträger
7	ungültige Speicherkontrollblöcke	27	Sektor nicht gefunden
8	zu wenig Speicherplatz	28	Drucker hat kein Papier mehr
9	ungültige Speicherblockadresse	29	Schreibfehler
10	ungültige SET-Kommandostrings („Umgebung")	30	Lesefehler
11	ungültiges Format (wovon, verrät uns DOS nicht)	31	Allgemeiner Fehler
12	ungültiger Dateizugriffscode	32	Datei-Sharing-Fehler
13	ungültige Daten	33	Datei-Locking-Fehler
14	Reserviert	34	unerlaubter Diskettenwechsel
15	ungültige Laufwerksspezifikation	35	kein Dateikontrollblock (FCB) verfügbar
16	Versuch, das aktuelle Verzeichnis zu löschen	80	Datei existiert bereits
17	anderes Gerät	81	Reserviert
18	keine weiteren Dateien vorhanden	82	nicht durchführbar
19	Diskette schreibgeschützt	83	Problem bei Interruptbearbeitung eines fatalen Fehlers

Tabelle 17-15 Die erweiterten Fehlercodes in Register AH nach Ausführung der Funktion 89 (hex 59)

Code	Beschreibung	Code	Beschreibung
1	Ressourcenknappheit (z. B. nicht genügend Platz)	8	gewünschte Sache (z. B. Datei) nicht gefunden
2	Zeitweilige Situation, späterer Versuch mag Erfolg haben	9	Falsches Format (z. B. unbekanntes Diskettenformat)
3	Autorisierung nicht gegeben (z. B. für Dateizugriff)	10	Gesperrt (z. B. Datei oder Datensatz)
4	Interner DOS-Fehler	11	Datenträger fehlen (z. B. CRC-Fehler auf einer Diskette)
5	Hardwarefehler	12	Bereits existent (z. B. Datei)
6	Fehler in der Systemsoftware (DOS)	13	Fehlerart unbekannt
7	Fehler in der Anwendungssoftware		

Tabelle 17-16 Die Fehlerarten in Register BH nach Ausführung der Funktion 89 (hex 59)

Code	Beschreibung	Code	Beschreibung
1	Sofort nochmals versuchen	5	Programm sofort stoppen
2	Nach Wartezeit nochmals versuchen	6	Fehler ignorieren und weiterarbeiten
3	Benutzer um Abhilfe bitten (z. B. Diskettenwechsel); siehe auch Code 7	7	Nach Benutzereingriff nochmals versuchen (siehe auch Code 3)
4	Programm „sauber" beenden (Dateien schließen usw.)		

Tabelle 17-17 Die Fortführungscodes in Register BL nach Ausführung der Funktion 89 (hex 59)

Code	Beschreibung	Code	Beschreibung
1	Unbekannt	4	Fehler bei einem seriell anzusprechenden Gerät (z. B. Drucker)
2	Fehler bei einem blockweise anzusprechenden Gerät (z. B. Diskettenlaufwerk)	5	Fehler im RAM-Speicher
3	Reserviert		

Tabelle 17-18 Die Fehlerlokalisierungscodes in Register CH nach Ausführung der Funktion 89 (hex 59)

17.3.2 Funktion 90 (hex 5A): Temporäre Datei anlegen

Mit Funktion 90 können Sie eine Datei anlegen, ohne sich darüber Gedanken machen zu müssen, ob eine Datei des entsprechenden Namens im spezifizierten Verzeichnis schon existiert. Es wird von DOS ein im Verzeichnis noch nicht vorhandener Dateiname gewählt. Der gebräuchliche Begriff *temporäre Datei* ist insoweit irreführend, als die angelegte Datei genauso fest im Verzeichnis steht wie jede andere Datei auch. Temporäre Dateien werden nicht, wie man vermuten könnte, von DOS unter bestimmten Umständen automatisch gelöscht. Vielmehr bezieht sich *temporär* auf die geplante Anwendung von Funktion 90. Wenn ein Programm Daten zeitweilig auf Diskette oder Platte auslagert, um sie später wieder einzulesen, ist der Dateiname für den Programmierer ohne Bedeutung. Nur in solchen Fällen sollte man DOS die Wahl des Dateinamens überlassen.
Zwei Parameter müssen vor dem Funktionsaufruf bereitgestellt werden: das Dateiattribut in Register CX und der Pfadname zu dem Verzeichnis, in dem die Datei angelegt werden soll, als ASCIIZ-String, auf den das Registerpaar DS:DX zeigt. Wird kein Pfad angegeben, nimmt DOS das aktuelle Verzeichnis des Standardlaufwerkes.
Der String, der den Pfadnamen enthält, muß so ausgelegt sein, daß er den von DOS gewählten Dateinamen der angelegten Datei aufnehmen kann. Das bedeutet, der Pfadname muß mit einem Schrägstrich (/) oder einem umgekehrten Schrägstrich (\) enden und es müssen 12 Bytes für den neuen Dateinamen frei sein.

Tritt bei der Operation ein Fehler auf, wird die Carry-Flagge gesetzt und der Fehlercode in Register AX gemeldet. Der Dateiname wird an den Pfadnamen im vorbereiteten String angefügt.

17.3.3 Funktion 91 (hex 5B): Neue Datei anlegen

Funktion 91 arbeitet ähnlich wie Funktion 60 (hex 3C), die (nicht unbedingt treffend) als *Datei anlegen* bezeichnet wird. Funktion 60 dient im Grunde dazu, eine bereits existierende Datei zu öffnen und nur sozusagen "im Notfall", wenn noch keine Datei vorhanden ist, eine anzulegen. Anders Funktion 91: Mit ihr lassen sich ausschließlich neue Dateien anlegen. Existiert bereits eine Datei des betreffenden Namens, kommt es zu einer Fehlermeldung.
Wie bei Funktion 60 werden die Dateiattribute in Register CX spezifiziert. DS:DX zeigt auf den ASCIIZ-String, der den Pfad- und den Dateinamen enthält. Im Falle eines Fehlers wird die Carry-Flagge gesetzt und ein Fehlercode in AX gemeldet.
Es gibt viele Fälle, bei denen man einen Standarddateinamen verwendet, um entweder eine bereits existierende Datei zu öffnen oder, falls keine passende Datei vorhanden ist, eine neue dieses Namens anzulegen. Für diese Zwecke wurde Funktion 60 in DOS aufgenommen. Es sind aber auch Situationen vorstellbar, in denen man eine bereits vorhandene Datei nicht zerstören und nur eine neue Datei anlegen möchte, wenn eine Datei dieses Namens noch nicht existiert. Dazu dient Funktion 91.

17.3.4 Funktion 92 (hex 5C): Dateizugriff sperren/freigeben

Funktion 92 ist im Zusammenhang mit Datei-Sharing sehr wichtig. Sie wird verwendet, um einen Teil einer Datei, auf den ein Programm zugreift, vor dem gleichzeitigen Zugriff anderer Programme zu schützen. Dadurch lassen sich gegenseitige Störungen vermeiden. Angenommen, ein Programm ändert zu einem bestimmten Zeitpunkt einen Datensatz. Dann darf natürlich nicht zum gleichen Zeitpunkt ein anderes Programm diesen Datensatz lesen wollen.
Die Zugriffssperre oder -freigabe einer Datei bzw. eines Teils davon wird über sechs Parameter festgelegt. AL gibt an, ob gesperrt (AL = 0) oder freigegeben (AL = 1) wird. BX enthält die Dateinummer. CX und DX werden zusammen als 4-byte-Ganzzahl behandelt und spezifizieren den Byte-Offset des gesperrten bzw. zu sperrenden Teils der Datei. SI und DI werden ebenfalls als 4-byte-Ganzzahl behandelt und enthalten die Länge des gesperrten oder zu sperrenden Teils. Das jeweils erste Register der Paare (also CX oder SI) enthält den höherwertigen Teil der Ganzzahl.
Datei-Sharing ist ein komplexer Prozeß. Daher empfiehlt es sich, die Regeln sehr streng einzuhalten. So sollten etwa Dateiteile nur auf die gleiche

Weise freigegeben werden, wie sie gesperrt wurden: Sind beispielsweise die ersten drei Datensätze mit *einer* Operation gesperrt worden, ist es nicht ratsam, sie mit drei Operationen freizugeben. Bevor eine Datei geschlossen wird, sollten alle Teile freigegeben werden, da der Schließvorgang dies nicht automatisch vornimmt.

17.3.5 Funktion 98 (hex 62): PSP-Adresse feststellen

Funktion 98 meldet die Adresse des Programmsegmentpräfixes als Segmentadresse in BX.
Früher war es üblich, daß DOS das PSP in die ersten 256 Bytes des Codesegmentes lädt. Dadurch war die Segmentadresse des PSP mit dem Inhalt des Codesegmentregisters identisch. Mit dem Aufkommen komplexerer PCs und neuer DOS-Versionen ist es komplizierter geworden. So werden beispielsweise im geschütztem Modus des 80286 im AT die Segmentregister auf eine neue und sehr unkonventionelle Weise verarbeitet. Routine 98 bietet eine Möglichkeit, die PSP-Adresse auch in Zukunft (hoffentlich!) unabhängig von der DOS-Version festzustellen.

Kapitel 18
Zusammenfassung: DOS-Routinen

18.1 Kurzzusammenfassung 310
18.2 Ausführliche Zusammenfassung 312

In diesem Kapitel finden Sie eine Auflistung aller DOS-Routinen. Aus der Kurzzusammenfassung können Sie ersehen, *welche* Routinen zur Verfügung stehen. Der ausführlichen Zusammenfassung ist zu entnehmen, *wie* die Routinen verwendet werden können. Genauere Erläuterungen zu allen DOS-Interrupts und DOS-Funktionen finden Sie in den Kapiteln 15, 16 und 17. Die nachfolgenden Zusammenfassungen dürften nur verständlich sein, wenn Sie die vorangegangenen drei Kapitel über DOS durchgearbeitet haben.

18.1 Kurzzusammenfassung

In DOS stehen neun Interrupts zur Verfügung, die fünf Hauptinterrupts (32, 37, 38, 39 und 47) sind in Tabelle 18-1 aufgelistet. Die verbleibenden vier Interrupts wurden nicht aufgenommen, da sie speziellen Zwecken dienen. Interrupt 33 (hex 21) wird für Funktionsaufrufe verwendet, die Interrupts 34, 35 und 36 sind Adreßinterrupts (siehe Kapitel 15).

Alle traditionellen DOS-Funktionen werden über Interrupt 33 (hex 21) aufgerufen, wobei die Funktionsnummer in Register AH stehen muß. Der Hauptvorteil der traditionellen DOS-Funktionen ist, daß sie in allen bis heute bekannten DOS-Versionen verfügbar sind. Sie sind in Kapitel 16 näher beschrieben.

Die neuen, erweiterten DOS-Funktionen wurden erstmals in DOS 2.00 implementiert, sie stehen in DOS Versionen 2 und 3 zur Verfügung. Aufgerufen werden sie mit der Funktionsnummer in Register AH über Interrupt 35 (hex 23). Mit DOS 3.00 wurden einige zusätzliche Funktionen eingeführt. Kapitel 17 ist den erweiterten DOS-Funktionen gewidmet.

In den nachfolgenden Tabellen sind die DOS-Funktionen aller DOS-Versionen bis einschließlich Version 3.00 enthalten.

Interrupt Dez	Hex	Erklärung
32	20	Programm beenden
37	25	Diskette/Platte: absolutes Lesen
38	26	Diskette/Platte: absolutes Schreiben
39	27	Beenden und im Speicher verbleiben
47	2F	Steuerung des Druckerspoolers (ab DOS 3.00)

Tabelle 18-1 Die fünf DOS-Hauptinterrupts

Funktion Dez	Hex	Erklärung	Funktion Dez	Hex	Erklärung
0	0	Programm beenden	22	16	Datei anlegen
1	1	Tastatureingabe mit Echo	23	17	Datei umbenennen
2	2	Bildschirmausgabe	25	19	Standardlaufwerk feststellen
3	3	Serielle Eingabe	26	1A	Diskettentransferbereich (DTA) festlegen
4	4	Serielle Ausgabe	27	1B	FAT-Informationen des Standardlaufwerkes lesen
5	5	Druckerausgabe			
6	6	Direkte Tastatur/Bildschirm-Ein-/Ausgabe	28	1C	FAT-Informationen eines beliebigen Laufwerks lesen
7	7	Direkte Tastatureingabe ohne Echo			
8	8	Tastatureingabe ohne Echo	33	21	Datensatz wahlfrei lesen
9	9	Zeichenkette (String) darstellen	34	22	Datensatz wahlfrei schreiben
10	A	Gepufferte Tastatureingabe	35	23	Dateilänge feststellen
11	B	Tastatureingabestatus prüfen	36	24	Feld für wahlfreien Zugriff setzen
12	C	Tastaturpuffer löschen und Funktion ausführen	37	25	Interruptvektor setzen
13	D	Laufwerks-Reset	38	26	Programmsegment anlegen
14	E	Standardlaufwerk bestimmen	39	27	Datensätze wahlfrei lesen
15	F	Datei öffnen	40	28	Datensätze wahlfrei schreiben
16	10	Datei schließen	41	29	Dateiname durchsuchen
17	11	Nach erster passender Datei suchen	42	2A	Datum ablesen
18	12	Nach nächster passender Datei suchen	43	2B	Datum stellen
19	13	Datei löschen	44	2C	Tageszeit ablesen
20	14	Datensatz sequentiell lesen	45	2D	Tageszeit stellen
21	15	Datensatz sequentiell schreiben	46	2E	Diskettenschreibverifikation setzen

Tabelle 18-2 Die traditionellen DOS-Funktionen

Funktion Dez	Hex	Erklärung	Funktion Dez	Hex	Erklärung
47	2F	DTA-Adresse feststellen	70	46	Dateinummernduplizierung erzwingen (CDUP)
48	30	DOS-Version feststellen			
49	31	Erweitertes Beenden und im Speicher verbleiben (KEEP)	71	47	Aktuelles Verzeichnis melden
			72	48	Speicherbereich belegen
51	33	Ctrl-Break-Abfrage testen oder festlegen	73	49	Speicherbereich freigeben
53	35	Interruptvektor feststellen	74	4A	Größe des belegten Speicherbereichs verändern (SET BLOCK)
54	36	Freie Diskettenkapazität feststellen			
56	38	Landesabhängige Informationen lesen	75	4B	Programm laden und ausführen (EXEC)
57	39	Unterverzeichnis anlegen (MKDIR)	76	4C	Erweitertes Programm beenden
58	3A	Unterverzeichnis löschen (RMDIR)	77	4D	Statuscode des Unterprogramms melden
59	3B	Aktuelles Verzeichnis ändern (CHDIR)	78	4E	Dateisuche beginnen (FIND FIRST)
60	3C	Datei anlegen (CREAT)	79	4F	Dateisuche fortsetzen
61	3D	Datei öffnen	84	54	Verifizierstatus feststellen
62	3E	Datei schließen	86	56	Datei umbenennen
63	3F	Lesen (Datei oder Gerät)	87	57	Datum und Zeit für Datei stellen/ablesen
64	40	Schreiben (Datei oder Gerät)			
65	41	Datei löschen	*DOS-3.00-Funktionen:*		
66	42	Dateizeiger bewegen	89	59	Erweiterten Fehlercode melden
67	43	Dateiattribute festlegen/feststellen (CHMOD)	90	5A	Temporäre Datei anlegen
			91	5B	Neue Datei anlegen
68	44	Ein-/Ausgabesteuerung für Geräte (IOCTL)	92	5C	Dateizugriff sperren/freigeben
			98	62	PSP-Adresse feststellen
69	45	Dateinummer duplizieren (DUP)			

Tabelle 18-3 Die erweiterten DOS-Funktionen (ab DOS 2.00)

18.2 Ausführliche Zusammenfassung

Die nachfolgende Auflistung der DOS-Funktionen zeigt auf, welche Register für die Parameterübergabe zu und von den Routinen benutzt werden müssen. Außerdem ist angegeben, ab welcher DOS-Version die Funktionen eingesetzt werden können:

DOS1	in allen DOS-Versionen verfügbar;
DOS2	ab DOS 2.00 verfügbar;
DOS3	ab DOS 3.00 verfügbar.

Routine	Funktion (hex)	Register Eingabe	Register Ausgabe	DOS-Version
Programmfunktionen				
Programm beenden	0	AH = 00		DOS1
Programmsegment anlegen	26	AH = 26 DX = Segmentadresse		DOS1
Erweitertes Beenden und im Speicher verbleiben	31	AH = 31 AL = Statuscode DX = Segmentadresse des ersten zu löschenden Segmentes	AX = Statuscode	DOS2
Ctrl-Break-Abfrage testen oder festlegen	33	AH = 33 AL = 00 (testen) AL = 01 (festlegen) DL = Festlegungscode	AX = Statuscode DL = Ctrl-Break-Status; 00 = inaktiv 01 = aktiv	DOS2
Programm laden und ausführen	4B	AH = 4B AL = Unterfunktionscode DS:DX = Zeiger auf ASCII-String ES:BX = Zeiger auf Kontrollblock	AX = Statuscode	DOS2
Erweitertes Programm beenden	4C	AH = 4C AL = Rückgabecode		DOS2
Statuscode des Unterprogramms melden	4D	AH = 4D	AX = Statuscode	DOS2
PSP-Adresse feststellen	62	AH = 62	BX = Segmentadresse des PSP	DOS3
Tastaturfunktionen				
Tastatureingabe mit Echo	1	AH = 01	AL = eingegebenes Zeichen	DOS1
Direkte Tastatureingabe ohne Echo	7	AH = 07	AL = eingegebenes Zeichen	DOS1
Tastatureingabe ohne Echo	8	AH = 08	AL = eingegebenes Zeichen	DOS1

Routine	Funktion (hex)	Register Eingabe	Ausgabe	DOS-Version
Gepufferte Tastatur-eingabe	A	AH = 0A DS:DX = Zeiger auf Eingabe-puffer		DOS1
Tastatureingabestatus prüfen	B	AH = 0B	AL = FF falls Zeichen ansteht AL = 00 falls kein Zeichen vorhanden	DOS1
Tastaturpuffer löschen und Funktion ausführen	C	AH = 0C AL = Funktionsnummer (1, 6, 7, 8 oder A) DL = auszugebendes Zeichen	AL = eingegebenes Zeichen	DOS1
Bildschirmfunktionen				
Bildschirmausgabe	2	AH = 02 DL = auszugebendes Zeichen		DOS1
Zeichenkette (String) darstellen	9	AH = 09 DS:DX = Zeiger auf Ausgabestring		DOS1
Bildschirm-Tastatur-Doppelfunktionen				
Direkte Tastatur/Bild schirm-Ein-/Ausgabe	6	AH = 06 DL = FF: Eingabe; 00-FE: auszugebendes Zeichen	AL = eingegebenes Zeichen	DOS1
Verschiedene Ein-/Ausgabefunktionen				
Serielle Eingabe	3	AH = 03	AL = eingegebenes Zeichen	DOS1
Serielle Ausgabe	4	AH = 04 DL = auszugebendes Zeichen		DOS1
Druckerausgabe	5	AH = 05 DL = auszugebendes Zeichen		DOS1
Laufwerksfunktionen				
Laufwerks-Reset	D	AH = 0D		DOS1
Standardlaufwerk	E	AH = 0E DL = Laufwerksnummer	AL = Anzahl Laufwerke	DOS1
Standardlaufwerk feststellen	19	AH = 19	AL = Laufwerksnummer	DOS1
Diskettentransferbereich (DTA) festlegen	1A	AH = 1A DS:DX = Zeiger auf DTA		DOS1
FAT-Informationen des Standardlaufwerkes lesen	1B	AH = 18	AL = Sektoren/Belegungseinheit CX = Bytes/Sektor DX = Anzahl Belegungseinheiten DS:DX = Zeiger auf FAT-Format-Byte	DOS1
FAT-Informationen eines beliebigen Laufwerkes lesen	1C	AL = 1C DL = Laufwerksnummer	CX = Bytes/Sektor AL = Sektoren/Belegungseinheit DX = Anzahl/Belegungseinheiten DS:DX = Zeiger auf FAT-Format-Byte	DOS1

Routine	Funktion (hex)	Register Eingabe	Register Ausgabe	DOS-Version
Diskettenschreib-verifikation setzen	2E	AH = 2E AL = Schreibverifikation: 00 = inaktiv 01 = aktiv DL = 00		DOS1
DTA-Adresse feststellen	2F	AH = 2F	AX = Statuscode ES:BX = Zeiger auf DTA	DOS1
Freie Disketten-kapazität feststellen	36	AH = 36 DL = Laufwerksnummer	AX = Sektoren/Cluster BX = Anzahl verfügbarer Cluster CX = Bytes/Sektor DX = Gesamtanzahl Cluster	DOS2
Verifizierstatus feststellen	54	AH = 54	AL = Verifizierstatus: 00 = inaktiv; 01 = aktiv	DOS2
Datei-Ein-/Ausgabefunktionen				
Datei öffnen	0F	AH = 0F DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Datei schließen	10	AH = 10 DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Nach erster passender Datei suchen	11	AH = 11 DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Nach nächster passender Datei suchen	12	AH = 12 DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Datei löschen	13	AH = 13 DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Datensatz sequentiell lesen	14	AH = 14 DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Datensatz sequentiell schreiben	15	AH = 15 DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Datei anlegen	16	AH = 16 DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Datei umbenennen	17	AH = 17 DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Datensatz wahlfrei lesen	21	AH = 21 DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Datensatz wahlfrei schreiben	22	AH = 22 DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Dateilänge feststellen	23	AH = 23 DS:DX = Zeiger auf FCB	AL = Statuscode	DOS1
Feld für wahlfreien Zugriff setzen	24	AH = 24 DS:DX = Zeiger auf FCB		DOS1

Routine	Funktion (hex)	Register Eingabe	Ausgabe	DOS-Version
Datensätze wahlfrei lesen	27	AH = 27 CX = Anzahl Datensätze DS:DX = Zeiger auf FCB	AL = Statuscode CX = Anzahl gelesener Datensätze	DOS1
Datensätze wahlfrei schreiben	28	AH = 28 CX = Anzahl Datensätze DS:DX = Zeiger auf FCB	CX = Anzahl geschriebener Datensätze	DOS1
Dateiname durchsuchen	29	AH = 29 DS:SL = Zeiger auf Kommandozeile ES:DI = Zeiger auf FCB AL = Suchspezifikationen (s. 16.1.38)	DS:SI = Zeiger auf Zeichen nach dem Dateinamen AL = Statuscode ES:DI = Zeiger auf FCB	DOS1
Datei anlegen (CREAT)	3C	AH = 3C CX = Dateiattribute DS:DX = Zeiger auf ASCIIZ-String	AX = Dateinummer oder Fehlercode	DOS2
Datei öffnen	3D	AH = 3D AL = Zugriffscode (s. 17.2.12) DS:DX = Zeiger auf ASCIIZ-String	AX: Dateinummer oder Fehlercode	DOS2
Datei schließen	3E	AH = 3E BX = Dateinummer	AX = Fehlercode	DOS2
Lesen (Datei oder Gerät)	3F	AH = 3F BX = Dateinummer CX = Anzahl zu lesender Bytes DS:DX = Zeiger auf DTA-Puffer	AX = Anzahl gelesener Bytes oder Fehldercode	DOS2
Schreiben (Datei oder Gerät)	40	AH = 40 BX = Dateinummer CX = Anzahl zu schreibender Bytes DS:DX = Zeiger auf ASCIIZ-String	AX = Anzahl geschriebener Bytes oder Fehlercode	DOS2
Datei löschen	41	AH = 41 DS:DX = Zeiger auf ASCIIZ-String	AX = Statuscode	DOS2
Dateizeiger bewegen	42	AH = 42 AL = Ausgangscode (s. 17.2.17) CX:DX = Offset-Wert	DS:AX = neue Zeigerposition (wenn CF = 0) X = Fehlercode (wenn CF = 1)	DOS2

Routine	Funktion (hex)	Register Eingabe	Ausgabe	DOS-Version
Dateiattribute festlegen/feststellen (CHMOD)	43	AH = 43 AL = 00: feststellen 01: festlegen CX = Attribute DS:DX = Zeiger auf ASCIIZ-String	AX = Statuscode CX = Attribute	DOS2
Dateinummer duplizieren (DUP)	45	AH = 45 BX = Dateinummer	AX = Dateinummer oder Fehlercode	DOS2
Dateinummern-duplizierung erzwingen (CDUP)	46	AH = 46 BX = erste Dateinummer CX = zweite Dateinummer	AX = Statuscode CX = Dateinummer	DOS2
Dateisuche beginnen (FIND FIRST)	4E	AH = 4E CX = Suchattribute DS:DX = Zeiger auf ASCIIZ-String	AX = Statuscode	DOS2
Dateisuche fortsetzen	4F	AH = 4F DS:DX = Informationen von Funktion 4E	AX = Statuscode	DOS2
Datei umbenennen	56	AH = 56 DS:DX = Zeiger auf ASCIIZ-String (alter Name) ES:DI = Zeiger auf ASCIIZ-String (neuer Name)	AX = erweiterten Standard-fehlercode	DOS2
Erweiterten Fehler-code melden	59	AH = 59 BX = 0000	AX = erweiterter Fehlercode BH = Fehlerartcode BL = Fortführungscode CH = Fehlerlokalisierungscode (s. 17.3.1)	DOS3
Temporäre Datei anlegen	5A	AH = 5A DS:DX = Zeiger auf Ver-zeichnispfadnamen CX = Dateiattribut	AX = Fehlercode, falls CF gesetzt DS:DX = Pfadname, falls CF nicht gesetzt	DOS3
Neue Datei anlegen	5B	AH = 5B DS:DX = Zeiger auf Datei-pfadnamen CX = Dateiattribut	AX = Dateinummer oder Fehlercode	DOS3
Dateizugriff sperren/ freigeben	5C	AH = 5C AL = 0 (sperren); 1 (freigeben) BX = Dateinummer CX:DX = Offset SI:DI = Anzahl Bytes	AX = Fehlercode	DOS3

Routine	Funktion (hex)	Register Eingabe	Register Ausgabe	DOS-Version
Verzeichnisfunktionen				
Unterverzeichnis anlegen (MKDIR)	39	AH = 39 DS:DX = Zeiger auf ASCIIZ-Code	AX = Statuscode	DOS2
Unterverzeichnis löschen (RMDIR)	3A	AH = 3A DS:DX = Zeiger auf ASCIIZ-String	AX = Statuscode	DOS2
Aktuelles Verzeichnis ändern (CHDIR)	3B	AH = 3B DS:DX = Zeiger auf ASCIIZ-String	AX = Statuscode	DOS2
Aktuelles Verzeichnis melden	47	AH = 47 DL = Laufwerksnummer DS:SI = Zeiger auf Datenbereich	AX = Fehlercode, wenn CF gesetzt DS:SI = Pfadname, wenn CF nicht gesetzt	
Datum/Zeit-Funktionen				
Datum ablesen	2A	AH = 2A	AL = Wochentag (0-6): 0 = Sonntag 6 = Samstag CS = Jahr (1980–2099) DH = Monat DL = Tag	DOS1
Datum stellen	2B	AH = 2B CX = Jahr (1980–2099) DH = Monat DL = Tag	AL = Statuscode	DOS1
Tageszeit ablesen	2C	AH = 2C	CL = Minuten CH = Stunden DL = Hundertstel Sekunden DH = Sekunden	DOS1
Tageszeit stellen	2D	AH = 2D CL = Minuten CH = Stunden DL = Hundertstel Sekunden DH = Sekunden	AL = Statuscode	DOS1
Datum und Zeit für Datei stellen/ablesen	57	AH = 57 AL = stellen/lesen: 00 = lesen 01 = stellen BX = Dateinummer CX = Zeit, wenn AL = 01 DX = Datum, wenn AL = 01	AX = erweiterter Fehlercode CX = Zeit, wenn AL = 00 DX = Datum, wenn AL = 00	DOS2

Routine	Funktion (hex)	Register Eingabe	Ausgabe	DOS-Version
Verschiedene Funktionen				
Interruptvektor setzen	25	AH = 25 AL = Interruptnummer DS:DX = Interruptvektor		DOS1
DOS-Version feststellen	30	AH = 30	AL = Nummer vor dem Punkt (z. B. *2*.1) AH = Nummer hinter dem Punkt (z. B. 2.*1*) BX = 0000 CX = 0000	DOS1
Interruptvektor feststellen	35	AH = 35 AL = Interruptnummer	AX = Statuscode ES:BX = Interruptvektor	DOS2
Landesabhängige Informationen lesen	38	AH = 38 DS:DX = Zeiger auf 32-byte-Bereich AL = 00 für Standardinformationen oder AL = Landescode oder AL = FF, wenn Landescode > 255 BX = Landescode, wenn AL = FF	AX = Statuscode DS:DX = Informationen (s. 17.2.7)	DOS2
Ein-/Ausgabesteuerung für Geräte (IOCTL)	44	AH = 44 AL = Code der Unterfunktion (s. 17.2.19) BL = Laufwerksnummer BX = Dateinummer CX = Anzahl der zu lesenden oder zu schreibenden Bytes	AX = Fehlercode, wenn CF gesetzt ist AX = Anzahl gelesener oder geschriebener Bytes DX = Geräteinformationen	DOS2
Speicherfunktion				
Speicherbereich belegen	48	AH = 48 BX = Anzahl benötigter Speichersegmente	AX = Segmentadresse oder Fehlercode, wenn CF gesetzt ist BX = Größe des längsten zusammenhängenden freien Speicherblocks	DOS2
Speicherbereich freigeben	49	AH = 49 ES = Segmentadresse des freizugebenden Bereichs	AX = Statuscode	DOS2
Größe des belegten Speicherbereichs verändern (SETBLOCK)	4A	AH = 4A BX = Neue Speichergröße in Segmenteinheiten ES = Segmentadresse des zu verändernden Bereichs	AX = Statuscode BX = maximale Größe (in Segmenten)	DOS2

Kapitel 19
Erstellen eines Programmes

19.1 Programmschnittstellen 320
19.2 Verknüpfen von Programmodulen 323
19.2.1 Schritt 1: Erstellen des Quellcodes 323
19.2.2 Schritt 2: Übersetzen des Quellcodes 323
19.2.3 Schritt 3: Binden von Programmen 324
19.2.4 Schritt 4: Umwandeln von Dateiformaten 324
19.2.5 Schritt 5: Anlegen von Objektcode-Bibliotheken 325
19.3 Das DOS-LINK-Programm 327
19.3.1 Binden eines einzelnen Programmes 327
19.3.2 Binden eines Programmes an die Compiler-Bibliothek 327
19.3.3 Verbinden von Programmen 328

Gleich zu Anfang des Kapitels sei ein Ratschlag wiederholt, der in diesem Buch schon des öfteren zu lesen war: Programmieren Sie stets auf der höchsten Ebene, auf der sich ein Softwareproblem lösen läßt. Nehmen Sie wenn möglich höhere Programmiersprachen wie BASIC, Pascal oder C in Anspruch. Wenn es sich als nötig erweisen sollte, setzen Sie einzelne DOS-Routinen (besser) oder BIOS-Routinen (schlechter) ein. Diese Tips sind gut - aber wenn Sie ein aktiver Programmierer sind, wissen Sie, daß man bei bestimmten Anwendungen einfach nicht umhinkommt, eigene Assemblerroutinen zu schreiben.
Wenn Sie Programme in den Grenzen einer einzigen Programmiersprache schreiben, genügen die Informationen, die üblicherweise in den Handbüchern zu finden sind, vollkommen. Anders sieht es aus, wenn auf eine Systemroutine zugegriffen oder ein Programmteil in einer anderen Programmiersprache eingebunden werden soll. In diesen Fällen müssen Sie viele technische Details des Betriebssystems und der Programmiersprache(n) kennen. Zum einen muß man wissen, wie DOS Programme bindet, das heißt, lauffähig macht, und zum anderen muß man in der Lage sein, Schnittstellen zwischen den Programmiersprachen herzustellen. In diesem Kapitel finden Sie Anmerkungen zu beiden Aspekten, die für die meisten Programmiersprachen Gültigkeit haben. Es wird vorausgesetzt, daß Sie mit den Methoden fortgeschrittener Programmierung vertraut sind.
In Kapitel 20 gehen wir gezielt auf fünf Programmiersprachen ein: Assembler, Interpreter-BASIC, Compiler-BASIC, Pascal und C. Dadurch sollen die allgemeinen Hinweise aus Kapitel 19 konkret umgesetzt werden. Hier sind nicht nur die reinen Sprachen von Bedeutung, sondern auch die Implementierungen, also die zugrundeliegenden Interpreter und Compiler.

19.1 Programmschnittstellen

Unter einer *Programmschnittstelle* wollen wir den Übergang von einer Programmiersprache zu einer anderen verstehen. Das ist ein weitläufiges Thema, wir werden uns nur mit Schnittstellen zwischen höheren Programmiersprachen einerseits und Assemblerroutinen andererseits befassen. Eine solche Schnittstelle besteht aus zwei Hauptteilen: der Kontrollschnittstelle und der Datenschnittstelle.
Die Kontrollschnittstelle dient dazu, die Kontrolle über den Programmablauf von einem Modul (z.B. der höheren Programmiersprache) zu einem anderen Modul (z.B. der Assemblerroutine) zu übertragen. Wenn man weiß, wie die Kontrollschnittstelle auszusehen hat, kann sie sehr einfach gehalten sein, der kleinste Fehler führt aber im allgemeinen dazu, daß nichts mehr funktioniert.
Die Datenschnittstelle hat die Aufgabe, Parameter von einem Modul auf ein anderes zu übertragen. Dabei müssen nicht nur die Inhalte, sondern auch die Datenformate beider Seiten berücksichtigt werden. Einzelheiten dazu finden Sie im nächsten Kapitel.

Wenn man Programme bzw. Programmteile zusammenbinden will, ist das Entwerfen von Schnittstellen nur ein Teil der dazu notwendigen Arbeit. Alle drei Komponenten, das aufrufende Programm, die aufgerufene Unterroutine und die Programmschnittstelle, müssen folgende Anforderungen erfüllen:

Das Programm muß in der Lage sein, den Weg zur Unterroutine zu finden. Systeme auf Basis des Prozessors 8088 können Unterroutinen auf zwei Arten aufrufen: über einen Interrupt oder über den Befehl CALL. Die DOS- und BIOS-Routinen werden durch Interrupts (Befehl INT) aufgerufen. Die Adressen der Unterroutinen sind implizit in den Interruptnummern enthalten. Viele der normalen Unterroutinen auf Assemblerbasis werden durch den Befehl CALL von der Programmiersprache aus aufgerufen. Die Adressen werden mit dem Programmnamen oder dem Namen der Unterroutine verbunden und während des Bindevorgangs festgelegt.

Es gibt zwei Sorten von CALL-Befehlen: FAR CALL und NEAR CALL. NEAR CALL bezieht sich auf eine Unterroutine im aktuellen Codesegment (CS) und erfordert keine Änderung des Registers CS. FAR CALL hingegen bewirkt den Aufruf einer Unterroutine außerhalb des aktuellen Codesegmentes und erfordert eine komplette segmentierte Adresse, wobei das CS-Register verändert werden muß. Einige Programmiersprachen verwenden beide Anweisungen, andere nur eine.

Die Unterroutine muß wissen, was nach der Abarbeitung zu tun ist. Meistens wird die Ablaufkontrolle nach Ausführung einer Unterroutine an das Programm, von dem sie aufgerufen wurde, zurückgegeben, entweder mit FAR RET oder mit NEAR RET. Natürlich kann es auch vorkommen, daß das Programm abgebrochen und die Kontrolle an DOS zurückgegeben werden soll.

Die Unterroutine muß wissen, welche Betriebsumgebung vom aufrufenden Programm hinterlassen wird. Zur Betriebsumgebung gehören beispielsweise die Inhalte der Segmentregister oder der Zustand des Stapels. Im allgemeinen wird CS das Codesegment enthalten und DS auf den Datenbereich des aufrufenden Programms zeigen. SS und PS werden üblicherweise vom Stapel des aufrufenden Programms gesteuert.

Die aufgerufene Unterroutine kann zumeist den Stapel des aufrufenden Programms mitverwenden, sofern kein übermäßig großer Stapel, sagen wir über 64 Bytes, benötigt wird. Grundsätzlich kann die Routine aber auch einen eigenen Stapel einrichten.

Wenn das Programm Informationen (Parameter) an die Unterroutine übergeben soll, müssen Programm und Unterroutine wissen, wo und wieviele Informationen abgelegt werden und ob sie geändert und zurückgegeben werden müssen. Meist arbeiten Programme mit einer festgelegten Anzahl von Parametern. Es gibt aber auch Programmiersprachen, die eine variable Anzahl erlauben. Die Parameter werden immer über den Stapel übergeben, direkt oder indirekt. Bei der direkten Methode wird der Parameterwert selbst über den Stapel geleitet, bei der indirekten Methode steht nur die Adresse des Parameters im Stapel.

Ob man direkte oder indirekte Übergabe wählt, ist hauptsächlich von der Programmiersprache abhängig. In einigen Sprachen steht nur eine Methode zur Verfügung, in anderen hat man die Wahl zwischen direkter und indirekter Parameterübergabe. Wollen wir z.B. eine Veränderung der Parameter durch die aufgerufene Routine vermeiden, geben wir die Parameterwerte direkt in den Stapel, so daß eine Kopie der ursprünglichen Werte erhalten bleibt. Sollen die Parameter von der Unterroutine geändert werden, übergeben wir die Adressen der Parameter und die Unterroutine kann die Parameterwerte verändern, indem sie über die im Stapel liegenden Adressen auf die Parameter zugreift.
Die Parameterübergabe ist der komplizierteste Teil einer Schnittstellenroutine. Sie ist bei den einzelnen Programmiersprachen je nach Datenhandhabung und Stapelverwaltung unterschiedlich schwer. Die Parameterübergabe wird der Hauptaspekt beim Vergleich der unterschiedlichen Programmiersprachen im nächsten Kapitel sein.
Die Unterroutine darf bestimmte Informationen nicht überschreiben. Bei aller Flexibilität, die ein Programmierer besitzen sollte, gibt es einige Grundregeln, die praktisch immer Gültigkeit haben. Einige der Regeln beziehen sich darauf, was man von Unterroutinen aus tun sollte und was nicht. Lesen Sie dazu auch Kapitel 2 für Hintergrundinformationen durch.
Man kann von einer Unterroutine aus Interrupts unterdrücken, sollte das aber nur ausnutzen, wenn es unbedingt notwendig ist (z.B. wenn Segmentregister geändert werden sollen). Es empfiehlt sich, alle Interrupts wieder zuzulassen, bevor die Kontrolle zum Hauptprogramm zurückgeht.
Wenn ein Segmentregister geändert wird, sollte der alte Wert auf dem Stapel gesichert werden. Ein anderes Register, dessen Ausgangswert auf jeden Fall erhaltenswert ist, ist der Basiszeiger (BS), der üblicherweise auf *die* Position im Stapel zeigt, an der die Parameter untergebracht sind. Die Arbeitsregister AX, BX, CX, DX, DI und SI können vom Unterprogramm beliebig verändert werden, ebenso die Flaggen. Nur in Ausnahmefällen sollte eines der Arbeitsregister zur Parameterübergabe herangezogen werden. Dazu steht der Stapel des Hauptprogramms zur Verfügung, der nicht zerstört werden sollte.
Der Stapel muß "gereinigt" werden, nachdem die Unterroutine abgeschlossen ist. Es gibt vier Dinge, die im Stapel stehen können, nachdem die Unterroutine abgeschlossen ist: die Parameter, die Rücksprungadresse des CALL-Befehls, einige Registerwerte des Hauptprogrammes und der Arbeitsbereich der Unterroutine.
Drei der Überbleibsel stellen kein Problem dar. Es gehört zur Aufgabe der Unterroutine, ihren Arbeitsbereich aus dem Stapel zu entfernen. Die CALL-Adresse wird von RET für den Rücksprung benötigt. Vor dem Aufruf gesicherte Register werden mit POP in das Hauptprogramm zurückgebracht. Schwierigkeiten bereiten im allgemeinen nur die Parameter, unter anderem, weil die Übertragung in unterschiedlichen Programmier-

sprachen auf verschiedene Weisen stattzufinden hat. Bei einigen Sprachen muß im RET-Befehl angegeben werden, wieviele Bytes zurückgegeben werden, bei anderen muß das Hauptprogramm die Werte holen. Die Unterschiede werden in Kapitel 20 ausführlich erläutert.
Mit diesen Überlegungen im Gedächtnis lassen Sie uns nun den kompletten Vorgang ansehen - von der Erstellung eines Programms oder einer Routine bis zu Verknüpfung mit anderen Programmen oder Routinen.

19.2 Verknüpfen von Programmodulen

In diesem Abschnitt wollen wir einige allgemein gültige Regeln für das Zusammenbinden von Programmen oder Programmodulen aufstellen. Wir verwenden eine DOS-Standardprogrammprozedur, die für alle Programmiersprachen, die im nächsten Kapitel angeführt sind, geeignet ist - mit Ausnahme des interpretierenden BASIC, das immer einen Sonderfall darzustellen scheint. Lassen Sie uns die Schritte ansehen, die zum Erstellen eines ablauffähigen Programmes notwendig sind.

19.2.1 Schritt 1: Erstellen des Quellcodes

Der erste Schritt beim Erstellen eines Programmes, nach der gedanklichen Vorarbeit, besteht darin, das Programm aus den Befehlen und Funktionen der Programmiersprache gemäß den syntaktischen Regeln aufzubauen. Der entstehende Programmtext wird *Quelltext* oder *Quellcode* genannt (*Source Code*). Für Programmiersprachen, die den DOS-Konventionen entsprechen, muß das Quellprogramm als ASCII-Textdatei (siehe Anhang C) vorliegen. Interpretierendes BASIC legt den Quelltext im allgemeinen nicht in einer ASCII-Textdatei ab, stellt diese Möglichkeit aber dennoch zur Verfügung (Option *A* des SAVE-Befehls).
Üblicherweise erhalten die Quellcodedateien eine Dateinamenerweiterung, die den Namen der Programmiersprache widerspiegelt, etwa BAS, PAS, ASM oder C.

19.2.2 Schritt 2: Übersetzen des Quellcodes

Ein Programm, das im Quellcode vorliegt, kann vom Computer nicht abgearbeitet werden. Bevor es ausgeführt werden kann, muß es von einem Übersetzer umgewandelt werden. Auf die Ausnahme des interpretierenden BASIC wird hier nicht näher eingegangen. Der Übersetzer wird im allgemeinen *Compiler* oder manchmal auch *Kompilierer* genannt. Eine Ausnahme bildet die Programmiersprache Assembler, bei der der Übersetzer *Assembler* heißt. Beachten Sie, daß das Wort *Assembler* einmal die Programmiersprache selbst, andererseits aber auch den Übersetzer für die

Sprache bezeichnet. Der Übersetzer (Compiler oder Assembler) setzt den Quellcode in Maschinensprachebefehle um, das Ergebnis wird *Objektcode* genannt. Auch im Objektcode kann das Programm noch nicht ausgeführt werden. Das Objektcodeformat ermöglicht es, daß mehrere Objektcodemodule zu einem einzigen größeren Programm zusammengefaßt werden können. Objektcodedateien haben vereinbarungsgemäß die Dateinamenerweiterung OBJ.

19.2.3 Schritt 3: Binden von Programmen

Der nächste grundlegende Schritt ist das *Binden* (*Linken*) von Programmen. Der Binder (Linker), in DOS LINK genannt, erfüllt zwei Hauptaufgaben: Er verknüpft zwei einzelne Objektmodule und er wandelt die Module bzw. das Gesamtprogramm vom Objektcode in das .EXE-Format um. Das Verknüpfen zweier Programmodule zu einer .EXE-Programmdatei ist ein so wichtiger Schritt bei der Programmerstellung, daß dies in einem eigenen Unterkapitel behandelt wird (Kapitel 19.3).
Die drei grundlegenden Schritte auf dem Weg zum ausführbaren Proggramm haben wir bereits angesprochen: das Erstellen des Quellcodes, das Kompilieren oder Assemblieren in den Objektcode und das Binden zu einem .EXE-Programm, das in den Speicher geladen und ausgeführt werden kann. Es gibt noch zwei Schritte, die zu dieser Konzeption gehören: das Umwandeln einer .EXE-Programmdatei, die mit LINK erstellt wurde, in das .COM-Format, und das Anlegen von Objektcode-Bibliotheken.

19.2.4 Schritt 4: Umwandeln von Dateiformaten

Programme werden im .EXE-Format auf Diskette oder Festplatte gespeichert, sind aber in dieser Form noch nicht zur Ausführung fertig. Daher muß DOS das Programm während des Ladevorgangs noch bearbeiten, so daß es ablauffähig wird. Es wird beispielsweise festgelegt, an welcher Speicherstelle das Programm beginnt und wie groß es ist. Außerdem muß ein Stapelbereich angelegt werden. Wenn die Betriebsbedingungen eines Programmes nicht zu kompliziert sind, können die genannten Vorbereitungen schon im voraus geschehen, indem das Programm in ein .COM-Dateiformat umgewandelt wird.
Eine .COM-Datei ist das genaue Abbild des Programmes, wie es im Computerspeicher steht. Während bei einer .EXE-Programmdatei noch diverse Vorbereitungen nötig sind, bevor das Programm ausgeführt werden kann, nimmt DOS bei einer .COM-Programmdatei nur zwei Aufgaben war: Das Programmsegmentpräfix PSP wird angelegt (siehe auch Kapitel 15.3) und die Segmentregister werden gesetzt.
Wir verwenden das DOS-Programm EXE2BIN, um eine .EXE-Datei in eine .COM-Datei umzuwandeln. Nicht alle Programme können in das

.COM-Format umgesetzt werden. Bei den geeigneten Programmen haben wir die Wahl, sie in der .EXE-Form zu belassen oder zu konvertieren. Beide Formen sind funktionell identisch, aber eine .COM-Datei ist kompakter und kann etwas schneller geladen werden.
Meistens hat man gar keine Wahlmöglichkeit zwischen .EXE und .COM, weil die Compiler der höheren Programmiersprachen häufig nur mit dem .EXE-Format arbeiten können. Das gilt z.B. für Compiler-BASIC oder Pascal. Nur Programme, die in Assembler geschrieben sind, können problemlos in das .COM-Format umgewandelt werden. Bei Programmen in C (Lattice/Microsoft C Compiler), einer Mittelsprache zwischen Assembler und höheren Programmiersprachen, dürfen beide Formate verwendet werden.
Sie können jederzeit herausfinden, ob ein Programm umgewandelt werden kann oder nicht. Versuchen Sie es einfach, und wenn es klappt - gut! Tritt eine Fehlermeldung auf, müssen Sie weiterhin im .EXE-Format arbeiten.

19.2.5 Schritt 5: Anlegen von Objektcode-Bibliotheken

Viele Compiler für höhere Programmiersprachen ziehen bei der Kompilierung Dutzende von Routinen zur Unterstützung heran, die in das zu kompilierende Programm eingebunden werden. Die Routinen liegen in Objektcodeform vor. Nun ist es sehr unbequem, aus so vielen Routinen diejenigen auszuwählen, die in ein Programm eingebunden werden sollen. Daher faßt man häufig mehrere Objektcoderoutinen in einer Datei zusammen und nennt diese Datei *Bibliothek* oder englisch *Library*. Die Dateinamenerweiterung lautet im allgemeinen LIB.
Zu den meisten Compilern höherer Programmiersprachen erhält man bereits beim Kauf eine fertige Bibliothek. Manche Compiler verfügen über verschiedene Bibliotheken, z.B. für unterschiedliche Betriebsumgebungen. So kann es beispielsweise Fließkommaroutinen geben, die einmal Gebrauch vom 8087 Arithmetik-Coprozessor machen und einmal nicht.
Der DOS-Binder (Linker) ist in der Lage, eine Bibliothek zu durchsuchen und die Routinen auszuwählen, die zur Vervollständigung eines Programmes benötigt werden. Ohne Bibliothek müßte der Programmierer selbst die Auswahl treffen. Wird eine Routine vergessen, endet der Bindeprozeß mit einem Fehler, werden zuviele Routinen angegeben, wächst das Programm auf eine unnötige Länge. Bibliotheken erleichtern die Arbeit beträchtlich.
Bei den meisten Programmiersprachen kommt man mit den Bibliotheken nicht direkt in Berührung. Vielmehr gibt es bestimmte Befehle oder Funktionen in der Programmiersprache, die nicht im Compiler-Kern enthalten sind, sondern für die eine Objektcodedatei in einer Bibliothek existiert. Wenn man den Befehl oder die Funktion im Quelltext einsetzt,

greift der Compiler bei der Kompilierung automatisch auf die Bibliothek zu und fügt dem Programm die Objektcoderoutine hinzu. Sie können aber auch mit dem DOS-Kommando LIB Objectcodedateien direkt aus einer Bibliothek auswählen.

Hinweis: Je nachdem, woher Sie Ihre DOS-Version bezogen haben (IBM-Handel, Kompatiblen-Hersteller, Softwarehaus etc.), finden Sie das LIB-Programm auf der Diskette oder nicht. LIB arbeitet mit einigen, aber nicht mit allen Compilern und Assemblern zusammen. Wenn Sie eine lauffähige LIB in die Hand bekommen - greifen Sie zu!

LIB kann verwendet werden, um bestehende Bibliotheken zu untersuchen. Sie sollten sich die Zeit dazu nehmen, da dies sehr interessant und lehrreich ist. Außerdem können Sie mit LIB Bibliotheksmodule ersetzen oder eigene Bibliotheken anlegen.
Nachfolgend finden Sie einige kleine Beispiele für den Umgang mit LIB. Um eine neue Bibliothek mit Namen TESTLIB anzulegen, geben Sie folgendes Kommando ein:

```
LIB TESTLIB;
```

Um die Inhalte einer bereits existierenden Bibliothek einzusehen, geben Sie

```
LIB TESTLIB,LTP1;
```

ein. Der Inhalt wird auf dem Gerät LTP1: (oder in eine Datei oder auf den Bildschirm) ausgegeben.
Wenn Sie der Bibliothek das Objektmodul X.OBJ hinzufügen wollen, geben Sie ein:

```
LIB TESTLIB+X;
```

Soll ein bereits existierendes Modul durch ein anderes ersetzt werden, schreiben Sie:

```
LIB TESTLIB-X+X;
```

Bei späteren LIB-Versionen geben Sie bitte -+X statt -X+X ein. Wollen Sie ein Modul aus der Bibliothek extrahieren, um es beispielsweise zu disassemblieren, geben Sie

```
LIB TESTLIB*X;
```

ein.
Die meisten Programme sind aus einer Reihe von Unterprogrammen oder Programmodulen zusammengesetzt. Ob Sie die Vorteile des LIB-Programms nutzen können, hängt davon ab, wie Sie Ihr Programm aufbauen. Geben Sie die Quellcodes aller Unterroutinen in eine Quelldatei und lassen Sie alle zusammen kompilieren, so ist das Programm LIB für Sie nur von geringer Bedeutung. Wenn Sie andererseits jede Unterroutine einzeln

kompilieren lassen, führt LIB genau *die* Aufgabe aus, die noch getan werden muß: Es faßt die Objektdateien zusammen. Welche Methode Sie bevorzugen, ist eine Frage des persönlichen Arbeitsstils. Die Wahl kann allerdings Konsequenzen haben, wenn man in Pascal programmiert. Lesen Sie hierzu Kapitel 20.4.

19.3 Das DOS-LINK-Programm

In Kapitel 19.2 wurde als dritter Schritt auf dem Weg zum lauffähigen Programm das *Binden* (*Linken*) genannt. Dazu steht das DOS-Kommando LINK zur Verfügung. Eine genaue Beschreibung von LINK finden Sie im DOS-Handbuch. Wir fassen hier kurz die wichtigsten und nützlichsten Operationen zusammen.
Das LINK-Programm nimmt vier Parameter in folgender Form an:

```
LINK 1,2,3,4;
```

Der erste Parameter ist eine Liste der Objektmodule (wie PROG1+PROG2+PROG3). Es folgt der Name, der dem fertigen Programm gegeben werden soll. Der dritte Parameter spezifiziert die Ausgabeeinheit für Meldungen (z.B. Bildschirm oder Drucker). Eine Liste der Bibliotheken, sofern welche benutzt werden sollen, bildet den vierten Parameter (z.B. BASCOM).

19.3.1 Binden eines einzelnen Programmes

Um ein vollkommen eigenständiges Programm, das als Objektcode vorliegt, zu binden, das heißt, in die .EXE-Form umzuwandeln, gibt man LINK und den Programmnamen ein. Nehmen wir beispielsweise das BEEP-Programm aus Kapitel 20.2.3, so lautet das Kommando:

```
LINK BEEP;
```

Dadurch entsteht ein Programm, das in den Speicher geladen und ausgeführt werden kann.

19.3.2 Binden eines Programmes an die Compiler-Bibliothek

Nehmen wir an, wir haben ein Programm in einer Programmiersprache wie kompiliertem BASIC geschrieben und wollen es an die Standard-Bibliothek anbinden, das heißt, in die Bibliothek aufnehmen. Der Name des Programmes sei X und die Compiler-Bibliothek soll BASCOM.LIB heißen. Der LINK-Befehl würde wie folgt aussehen:

```
LINK X,,BASCOM;
```

Normalerweise erstellt der Compiler ein Objektmodul, das alle benötigten Teile aus der Bibliothek selbst aufruft. Im Falle des Lattice/Microsoft C-Compilers, der in Kapitel 20.5 eingehend besprochen wird, gibt es eine Startroutine (*Präfixmodul*), die an den Anfang jedes Programmes angebunden werden muß. Nehmen wir an, das Präfixmodul besitzt den Namen C, unser Programm heißt X und die angesprochene Bibliothek trägt den Namen MC, dann wäre folgender Befehl nötig:

```
LINK C+X,X,,MC
```

In diesem Beispiel gibt es zwei neue Punkte zu beachten. Erstens wurde das LINK-Programm explizit aufgefordert, zwei Objektcodedateien zu verbinden, C.OBJ und X.OBJ (die Datei, die wir kompilierten) und zweitens haben wir der erstellten .EXE-Datei einen eigenen Namen zugewiesen, nämlich X. Wir müssen den Namen definieren, weil LINK andernfalls den Namen des ersten genannten Moduls, das wäre C, verwendet. Man kann dem umgesetzten Programm einen beliebigen Namen geben, üblicherweise läßt man aber den Dateinamen unverändert und ändert nur die Dateinamenerweiterung von .OBJ zu .EXE.

19.3.3 Verbinden von Programmen

Wie wissen, wie ein einzelnes Programm an eine Bibliothek gebunden wird. Nun sollen Sie lernen, wie zwei oder mehrere Programmodule mit LINK zusammengebunden werden. Gehen wir nochmal für einen Moment zum Anbinden an Bibliotheken zurück. Wenn Sie an eine Bibliothek BIB, die sowohl Pascal- als auch Assemblermodule enthält, das Pascal-Programm X anbinden wollen, geben Sie

```
LINK X,,PASCAL+BIB
```

Nun stellen Sie sich vor, Sie hätten keine Bibliothek angelegt und wollen nur zwei Objektdateien miteinander verbinden; der Inhalt der einen Datei wurde ursprünglich in einer höheren Programmiersprache geschrieben und kompiliert, der der anderen wurde vom Assembler mit DOS- und BIOS-Schnittstellen assembliert. Unsere Programmiersprache soll Pascal sein, das Programm heißt, wie immer, X und die Assemblerschnittstelle SCHNITT. In diesem Fall ist einzugeben:

```
LINK X+SCHNITT,,PASCAL
```

Es gibt, wie Sie sich sicherlich vorstellen können, unendliche viele Möglichkeiten, Programmodule miteinander zu kombinieren. Anhand der Erklärungen und Beispiele dieses Kapitels sollten Sie aber die Grundlagen des Umgangs mit LINK gelernt haben.

Kapitel 20
Programmiersprachen

20.1 Besonderheiten der einzelnen Programmiersprachen 331
20.2 Assembler 331
20.2.1 Aufbau einer Assemblerroutine 332
20.2.2 Schnittstellenkonventionen 333
20.2.3 Erstellen und Binden eines Assemblerprogramms 334
20.3 Interpretierendes und kompiliertes BASIC 336
20.3.1 BASIC-Datenformate 336
20.3.1.1 Ganzzahldatenformate 336
20.3.1.2 Fließkommadatenformate 338
20.3.1.3 Stringdatenformate in interpretierendem BASIC 339
20.3.1.4 Stringdatenformate in kompiliertem BASIC 340
20.3.2 Assemblerschnittstellen in interpretierendem BASIC 341
20.3.3 Assemblerschnittstellen in kompiliertem BASIC 344
20.4 Pascal 347
20.4.1 Pascal-Datenformate 348
20.4.1.1 Ganzzahldatenformate 348
20.4.1.2 Stringdatenformate 349
20.4.1.3 SET-Datenformat 350
20.4.1.4 Fließkommadatenformate 350
20.4.2 Assemblerschnittstellen in Pascal 352
20.5 Programmiersprache C 355
20.5.1 C-Datenformate 355
20.5.1.1 Ganzzahldatenformate 356
20.5.1.2 Stringdatenformate 356
20.5.1.3 Fließkommadatenformate 357
20.5.2 Assemblerschnittstellen in C 358
20.5.3 Parameterübergabe in C 359
20.6 Schlußbemerkung 360

Im letzten Kapitel haben wir uns damit beschäftigt, wie Programme erstellt und mit LINK lauffähig gemacht werden können. In diesen Kapitel geht es um die Programmiersprachen, in denen die Programme geschrieben werden. Nun reicht ein einziges Kapitel natürlich nicht aus, um die Grundlagen von vier Programmiersprachen darzustellen. Vielmehr soll uns die Frage beschäftigen, wie Module in höheren Programmiersprachen mit Assemblerroutinen (z.B. DOS- oder BIOS-Routinen, aber auch selbstgeschriebenen Routinen) verbunden werden können.
Wenn Sie sich diesem Problem zuwenden, werden Sie feststellen, daß die abstrakten Beschreibungen der Programmiersprachen, wie sie in vielen Büchern zu finden sind, sich als wenig hilfreich erweisen. Das liegt daran, daß es nicht genügt, eine Sprache "im großen und ganzen" zu beherrschen, sondern man muß auch die vielen Details kennen. Zu diesen Details zählen nicht nur die Sprachelemente, sondern auch die Implementierung und die Umgebung.
Jede Implementierung - ob Compiler oder Interpreter - bringt Besonderheiten mit sich, deren Kenntnis in der Programmierpraxis unerläßlich ist. Ebenso muß man über die Betriebssystemumgebung und das Computermodell Bescheid wissen, auf dem ein Programm erstellt und/oder ausgeführt werden soll. Gerade die meistgelobten Programmiersprachen wie etwa Pascal sind von der Sprachdefinition her so angelegt, daß man sie für professionelle Programme nur einsetzen kann, wenn man die Grenzen der Sprache verläßt und zum Teil auf unteren Ebenen (z.B. Assembler) arbeitet. Häufig werden bei der Implementierung der Sprache z.B. in einem Compiler bereits die Besonderheiten der jeweiligen Hardware berücksichtigt. Das führt dazu, daß jede Sprachimplementierung ein bißchen anders aussieht.
Von BASIC gibt es Hunderte von Dialekten. Wer BASIC als abstrakte Sprache gelernt hat, kann noch lange kein professionelles Programm in einer der BASIC-Versionen schreiben.
Für dieses Kapitel wurden die Sprachen Assembler, BASIC, Pascal und C ausgesucht. Die Implementierungen sind der IBM Macro Assembler, IBM Interpreter-BASIC, IBM Compiler-BASIC, IBM Pascal und Lattice/ Microsoft C. Damit sind die auf dem IBM PC am häufigsten eingesetzten Sprachen abgedeckt. Es ist empfehlenswert, andere Sprachimplementierungen nur nach sorgfältiger Abwägung zu verwenden. Sprachen, die mit DOS-LINK nicht kompatibel sind, sollten Sie grundsätzlich meiden. Leider sind recht viele Compiler auf dem Markt, die entweder gar keinen linkbaren Objektcode erzeugen, beispielsweise Borlands Turbo Pascal oder Logitechs Modula-2, oder die Objektcode in einem zu LINK inkompatiblen Format erzeugen, beispielsweise die Sprachimplementierungen von Digital Research. Wer mit solchen Compilern arbeitet, macht sich stark von den jeweiligen Herstellern abhängig. Das ist nur dann ratsam, wenn die DOS-inkompatible Implementierung überzeugende Vorteile bietet.

20.1 Besonderheiten der einzelnen Programmiersprachen

Die fünf Sprachimplementierungen, die im folgenden besprochen werden, sind in mehreren Versionen erhältlich, so daß man im Grunde von verschiedenen Implementierungen sprechen muß. Glücklicherweise sind die Unterschiede zwischen den einzelnen Versionen aber relativ gering, jedenfalls nicht so groß wie z.B. zwischen BASIC und Pascal.

Assembler. Die Erläuterungen zu Assembler stützen sich auf Version 1.00 des IBM Makro-Assemblers, der von Microsoft entwickelt wurde. Eine Reihe anderer Versionen ist von Microsoft, IBM und anderen Computerherstellern, die eine Lizenz für Microsofts Grundprodukt besitzen, erhältlich. Diese Versionen unterscheiden sich nur in einigen wenigen Aspekten, die uns hier nicht zu interessieren brauchen.

Interpretierendes BASIC. Das interpretierende BASIC, das wir in diesem Kapitel behandeln wollen, gibt es in Hunderten von Varianten mit mindestens ebensovielen Namen. IBM-Benutzern ist die Version, die wir hier besprechen, als BASIC oder BASICA bekannt, Versionsbezeichnungen sind C1.10, J1.00 oder A2.10. Außerhalb der IBM-Welt wird es als BASIC, Microsoft-BASIC oder GW-BASIC bezeichnet. Wir beschäftigen uns nur mit den gemeinsamen Elementen, die Unterschiede der einzelnen Versionen bleiben unberücksichtigt.

Kompiliertes BASIC. Für die Erklärungen zum kompilierten BASIC haben wir uns auf die Version 1.00 des IBM BASIC Compiler gestützt. Was hier gesagt wird, gilt prinzipiell auch für andere BASIC Compiler wie z.B. Business BASIC.

Pascal. Bei Pascal verwenden wir als Grundlage IBMs Version 1.00. Die Details, die erwähnt werden, gelten auch für die Version 2.00 und für verschiedene Microsoft-Versionen.

Die Sprache C. Hier beziehen wir uns auf den von Lattice entwickelten Lattice/Microsoft C-Compiler Version 1.04. Es gibt andere ähnliche Versionen von Microsoft, Lattice und Lifeboat, die auf der gleichen Grundlage basieren. Verwechseln Sie diesen Compiler nicht mit dem Microsoft C-Compiler (Version 3 und höher), der entwickelt wurde, nachdem die Lattice/Microsoft-Version bereits am Markt war.

20.2 Assembler

Es gibt zwei grundsätzlich verschiedene Arten von Assemblerprogrammen. Da sind einmal die Unterroutinen, die von anderen Programmen (die in höheren Programmiersprachen abgefaßt sein können) aufgerufen werden. Auf der anderen Seite gibt es eigenständige Assemblerprogramme. Die Routinen sind von der Unterstützung des aufrufenden Programmes weitgehend abhängig, dadurch wird auch ihre Struktur bestimmt. Eigenständige Programme hingegen müssen eine eigene Struktur haben und alle nöti-

gen Operationen völlig selbständig abwickeln. Assemblerunterroutinen sind im allgemeinen relativ einfach zu schreiben, während die Entwicklung eigenständiger Assemblerprogramme einiges Kopfzerbrechen bereiten kann. Als Schnittstelle zwischen BIOS- oder DOS-Systemroutinen und einer höheren Programmiersprache kommen nur Assemblerroutinen in Frage.
Im folgenden erhalten Sie Einblick in einige Programmiertechniken, die für die Entwicklung einer Schnittstellenroutine in Assembler wichtig sind. Wir werden außerdem den Entstehungsprozeß eines eigenständigen Assemblerprogrammes durchsprechen, ohne einen Einführungskurs in Assembler zu geben. Wenn Sie Assembler lernen möchten, empfiehlt es sich, nicht nur Lehrbücher zu lesen, sondern darüberhinaus kommentierte Assemblerlistings durchzuarbeiten. Damit profitieren Sie am meisten von den Erfahrungen anderer Programmierer. Sehr interessant ist beispielsweise das BIOS-Listing. Eine andere Quelle tut sich bei den meisten Compilern auf, sofern sie Assemberlistings ausgeben können. Der Vorteil bei Compiler-Listings: Indem Sie bestimmte Befehle der Programmiersprache kompilieren lassen, können sie überprüfen, wie sie in Assembler umgesetzt werden. Dadurch lernen Sie zugleich einiges über die Schnittstellenkonventionen für Unterroutinen, die der Compiler verwendet. Ein anderer, nicht so guter Weg, Assemblerkenntnisse zu vertiefen, besteht darin, Assemblerprogramme aus dem Speicher zu disassemblieren, also aus dem Maschinencode in Assemblercode rückzuübersetzen. In DOS steht dazu der Befehl U (*Unassemble*) des Kommandos DEBUG zur Verfügung. Um es einmal zu sagen: Der Autor hat den größten Teil seines Wissens über PC-Assemblerprogrammierung aus der Analyse von Programmen gewonnen. Natürlich muß man sich die Grundkenntnisse zuvor auf andere Weise angeeignet haben.

20.2.1 Aufbau einer Assemblerroutine

Die meisten Assemblerroutinen sind gleichartig aufgebaut. In Kapitel 8.4.1 haben wir die logische Struktur einer Schnittstellenroutine in fünf Ebenen eingeteilt:

Ebene 1: Vereinbarung der Assemblerroutine
  Ebene 2: Vorbereitung der Routine
    Ebene 4: Parameterübergabe zur Routine
      Ebene 5: Aufgabe erfüllen (z.B. DOS- oder BIOS-Routine aufrufen)
    Ebene 4: Parameterübergabe zum aufrufenden Programm
  Ebene 2: Nachspann der Routine
Ebene 1: Assemblernachspann

Der Aufbau gilt nicht nur für Schnittstellen-, sondern auch für die meisten anderen Assemblerroutinen. Die Kodierung ist natürlich von Fall zu Fall verschieben.

Das Standardwerkzeug zur Erstellung von Assemblerroutinen ist der Makro-Assembler von Microsoft, der kurz MASM genannt wird und in verschiedenen Versionen von unterschiedlichen Seiten erhältlich ist. Um die in diesem Buch beschriebenen Programmiertechniken nachzuvollziehen, genügt auch ein einfacherer Assembler als der MASM.

20.2.2 Schnittstellenkonventionen

Wenn Sie ein Modul in einer höheren Programmiersprache mit einer Assemblerroutine verbinden wollen, genügt es nicht, beide Sprachen zu beherrschen, sondern Sie müssen auch die Schnittstellenkonventionen kennen. Die Schnittstelle selbst besteht aus einem Assemblerprogramm, das zwischen dem Modul in der höheren Programmiersprache und der Assemblerroutine liegt. Aufgabe der Schnittstelle ist es, Kontrolle und Parameter vom einen zum anderen Programm und zurück zu übergeben. Dabei müssen die Besonderheiten der höheren Programmiersprache (z.B. Datenformate) berücksichtigt werden. Wenn diese nicht dokumentiert sind, bleibt Ihnen nichts anderes übrig, als sie dem Compiler zu entnehmen. Die Methode, vom Compiler während der Kompilierung ein Listing des Assemblercodes erzeugen zu lassen, wurde bereits erwähnt. Zu den meisten Compilern werden Assemblerroutinen mitgeliefert, die zu studieren sich im allgemeinen ebenfalls lohnt. Zudem können Sie die Routinen als Vorbild für eigene Assemblerentwicklungen nehmen und sich dadurch die Arbeit erleichtern.
Den einfachsten Zugriff erhält man auf die Routinen, die in den Compiler-Bibliotheken verwendet werden. Suchen Sie sich einen Funktionsbereich aus, z.B. Ein-/Ausgabe oder Bildschirmsteuerung, und stellen Sie fest, welche Routinen im betreffenden Modul aufgerufen werden. Dazu müssen Sie die Bibliotheksmodule einsehen. Beachten Sie, daß die Namen von Modulen und Routinen häufig nicht identisch sind. Mit dem DOS-Programm LIB läßt sich das Inhaltsverzeichnis einer Bibliothek auflisten (siehe Kapitel 19) und einer Datei zuordnen. Angenommen, Sie wollen die Bibliothek SPRACHE.LIB der Datei LISTING zuweisen, geben Sie ein:

```
LIB SPRACHE,LISTING;
```

Wenn Sie die Auflistung ansehen, finden Sie die Routine, die Sie interessiert und den Namen des Moduls, das die Routine enthält. Um das Modul abzutrennen und unter dem Namen XMOD anzulegen, geben Sie

```
LIB LANG*XMOD;
```

ein. Mit dem Stern veranlassen Sie, daß LIB eine Kopie des Moduls als eigenständige Objektdatei anfertigt, die im Beispiel den Namen XMOD.OBJ erhält.

Nun könnten Sie das Modul XMOD näher betrachten. Davon ist aber abzuraten, den es enthält noch Link-Editor-Informationen, die unnötig verwirren. Stattdessen sollten Sie XMOD.OBJ in zwei Schritten in ein reines Maschinenspracheprogramm umwandeln. Geben Sie dem Modul zunächst die Form einer .EXE-Datei:

```
LINK XMOD;
```

Es entsteht die Datei XMOD.EXE. Eventuelle Fehlermeldungen, daß kein Stapel vorhanden sei, brauchen Sie nicht zu beachten. Im zweiten Schritt wandeln Sie nun XMOD.EXE in eine .COM-Programmdatei um, so daß der .EXE-Vorspann wegfällt:

```
EXE2BIN XMOD.EXE XMOD.COM
```

Die entstehende Programmdatei XMOD.COM beinhaltet reinen Maschinencode, den Sie mit DEBUG in Assemblertext umwandeln und einsehen können. Stellen Sie die Größe von XMOD.COM fest und laden Sie das Programm mit DEBUG ein:

```
DEBUG XMOD.COM
```

Mit *Unassemble* läßt sich der Assemblercode auflisten:

```
U 100 L xxx
```

xxx bezeichnet die Länge von XMOD.COM in hex.
Der geschilderte Vorgang mag Ihnen aufwendig und unbequem vorkommen, in der Praxis sind die Schritte aber schnell erledigt. Der nächste Abschnitt enthält ein Beispiel.

20.2.3 Erstellen und Binden eines Assemblerprogramms

Wir wollen anhand eines einfachen Programms, das einen Ton erzeugt, besprechen, wie man ein Assemblerprogramm erstellt und ablauffähig macht. Das Programm ist insofern unsinnig, als mit CHR$(7) auf jedem PC ein Ton erzeugt werden kann; als Beispiel ist die Routine aber bestens geeignet.
Der Ablauf der Routine ist einfach: Mit der DOS-Funktion 2, die über Interrupt 33 aufgerufen wird, erzeugt das Programm einen Ton und gibt die Kontrolle über Interupt 32 an DOS zurück. Der Quellcode unseres Beispielprogramms sieht wie folgt aus:

```
;DOS Ton-Programm
TONSEG    SEGMENT BYTE PUBLIC 'PROC'
          ASSUME  CS:TONSEG
TON       PROC
          MOV     DL,7      ;Bell-Zeichen (Glocke, CHR$(7))
          MOV     AH,2      ;Zeichenausgabefunktion anwählen
          INT     33        ;DOS-Funktionsinterrupt aufrufen
          INT 32            ;Rückkehr zu DOS über Interupt 32
TON       ENDP
TONSEG    ENDS
          END
```

Wie Sie sehen, ist das Programm nur vier Instruktionen lang und belegt nur acht Bytes. Mit folgendem Befehl können wir das Programm assemblieren:

```
MASM TON;
```

Der MASM-Befehl legt eine Objektdatei an, die in ein ablauffähiges Programm umgewandelt werden kann. Das geschieht mit:

```
LINK TON;
```

Es entsteht die Programmdatei TON.EXE. LINK geht davon aus, daß im Programm ein Stapelsegment zu finden ist. Da das bei unserem einfachen Beispiel entfällt, tritt eine Fehlermeldung auf, die aber ohne Bedeutung ist. Nur ein Programm ohne Stapelsegment kann in eine .COM-Programmdatei konvertiert werden. Dazu nehmen wir EXE2BIN:

```
EXE2BIN TON.EXE TON.COM
```

Das Programm trägt nun den Namen TON.COM und ist auf jedem PC unter DOS ausführbar.
Beachten Sie, wie sich mit den einzelnen Schritten die Länge des Programms beträchtlich verändert. Der Quellcode beträgt ungefähr 400 Bytes (das hängt von der Länge der Kommentare ab). Wenn wir das Programm assemblieren lassen, entstehen genau acht Bytes Maschinencode. Dennoch besitzt die Objektdatei eine Länge von 54 Bytes, da noch einige Standard-LINK-Informationen mit abgespeichert werden. 54 Bytes ist zwar schon wesentlich weniger als die ursprünglichen 400 Bytes, aber noch deutlich mehr als die eigentlich interessanten acht Bytes. Nach dem Binden schwillt die 54 Bytes lange Objektdatei zu einer .EXE-Datei von 520 Bytes Länge an. Das .EXE-Format enthält einen Vorspann von 512 Bytes, der Informationen darüber enthält, wie das Programm zu laden ist. Bei der Umwandlung in das .COM-Format wird der Vorspann herausgenommen und es verbleiben die acht Bytes reiner Maschinencode.

20.3 Interpretierendes und kompiliertes BASIC

Um es vorweg zu sagen: Die Programmierung von Schnittstellen zwischen BASIC- und Assemblerprogrammen ist ein derart komplexes Thema, daß man damit mühelos mehrere Bücher füllen kann. Das liegt daran, daß BASIC nur sehr unsaubere Schnittstellen anbietet. Die Tatsache, daß es verschiedene BASIC-Versionen für unterschiedliche PC-Modelle gibt, kommt erschwerend hinzu.
Wir wollen uns auf die Kommunikations- und die Datenschnittstellen von BASIC beschränken. Beginnen wir mit den Datenformaten.

20.3.1 BASIC-Datenformate

BASIC kennt vier Datenformate: Ganze Zahlen (*Integer*), Strings variabler Länge, kurze Fließkommazahlen und lange Fließkommazahlen. Statt *kurz* und *lang* sagt man in BASIC *einfachgenau* und *doppeltgenau*. Um das Format einer Variablen festzulegen, wird dem Variablennamen ein Zeichen angehängt: % für Ganzzahl, ! für einfache Genauigkeit (kurze Fließkommazahlen), # für doppelte Genauigkeit (lange Fließkommazahlen) und $ für String. Numerische Konstanten können analog klassifiziert werden. Implizite Definitionen sind über DEF möglich, das Standardformat ist einfache Genauigkeit. Hier sind einige einfache Beispiele:

A%	Ganzzahlvariable (Integer)
A!	einfachgenaue Variable
A#	doppeltgenaue Variable
A$	Stringvariable
1%	Ganzzahlkonstante
1!	einfachgenaue Konstante
1#	doppeltgenaue Konstante
"1"	Stringkonstante

Wichtiger Hinweis: Während die drei numerischen Datenformate für interpretierendes und kompiliertes BASIC identisch sind, sieht das Stringformat für kompiliertes BASIC anders aus.

20.3.1.1 Ganzzahldatenformate

Das Standard-Ganzzahlformat besteht beim PC aus einem 16-bit- oder 2-byte-Wort mit einem Wertebereich von 0 bis 65.535. In Kapitel 2.1.1 finden Sie nähere Erläuterungen dazu. Das BASIC-Ganzzahlformat weicht davon ab, indem ein Bit als Vorzeichen dient und sich der Wertebereich daher von 32.768 über 0 bis +32.767 erstreckt. Da die vorzeichenlose 16-bit-Ganzzahl mit Werten von 0 bis 65.535 ein grundlegendes Element der Speicheradressierung und Adressenberechnung des PC darstellt, sind die

Probleme in BASIC schon vorprogrammiert. Es hat sich eingebürgert, in BASIC Adressen über 32.767 mit negativen Zahlen zu beschreiben (die Bitkodierungen von -32.768 und 65.535 sind identisch). Das ist auch der beste Weg, um Adressen an Assemblerschnittstellen zu übergeben. Wenn Sie vorzeichenlose Ganzzahlen, die in einem Format mit Vorzeichen abgelegt sind, auf dem Bildschirm ausgeben und zur Grundlage von Berechnungen nehmen wollen, ist allerdings Vorsicht geboten. Um Fehler zu vermeiden, sollten Sie Adreßrechnungen im doppeltgenauen Format (lange Fließkommazahlen) vornehmen.
Wenn Sie das Ganzzahlformat für vorzeichenlose Adressen verwenden, können Sie sich eines Tricks bedienen. Theoretisch kann eine Ganzzahl in BASIC keinen Wert annehmen, der größer als 32.767 ist. Einer Ganzzahlkonstanten kann also beispielsweise nicht der Wert 50.000 zugewiesen werden. In Hexadezimalschreibweise hingegen lassen sich Zahlen zwischen 0 (hex &H0) und 65.535 (hex &HFFFF) ausdrücken und zuordnen. Das bedeutet, daß BASIC *I%=50.000* nicht zuläßt, wohl aber das hexadezimale Äquivalent *I%=&HC350*. Da BASIC die Funktion HEX$ zur Umwandlung von Dezimal- in Hexadezimalzahlen zur Verfügung stellt, bietet es sich an, Adressen in Hexnotation zu schreiben. Berechnung können allerdings ausschließlich im Dezimalsystem vorgenommen werden.
Für Berechnungen kann eine Adresse vom Ganzzahl- in ein Fließkommaformat umgewandelt werden. Beispiel:

```
IF I% < 0 THEN D# = I% + 65535# ELSE D# = I%
```

I% ist eine Ganzzahlvariable, D# die doppeltgenaue Entsprechung. Um Adressen von doppelter Genauigkeit in ein Ganzzahlformat umzuwandeln, schreiben Sie folgende Zeile:

```
IF D# > 32767 THEN I% = D# - 65536 ELSE I% = D#
```

Die BASIC-Funktion VARPTR stellt die Offsetadresse einer Ganzzahlvariablen innerhalb des BASIC-Standarddatensegmentes zur Verfügung. Über die Adresse kann mit PEEK oder POKE auf den Variableninhalt zugegriffen werden. VARPTR ist auch bei Fließkomma- und Stringvariablen von Bedeutung, wie Sie noch sehen werden. Das folgende Beispiel bezieht sich auf Ganzzahlvariablen:

```
I% = 999 'beliebiger Wert
I.ZEIGER = VARPTR (I%)
J% = PEEK (I.ZEIGER) * 256 + PEEK (I.ZEIGER + 1)
PRINT I%, J%
```

In der Praxis hat VARPTR eine recht geringe Bedeutung. Das obige Beispiel ist aber gut geeignet, Ihr Verständnis der Speicheradressierung in BASIC zu fördern. Für die Entwicklung einer Assemblerschnittstelle ist es wichtig, mit diesem Aspekt vertraut zu sein.

20.3.1.2 Fließkommadatenformate

Fließkommazahlen sowohl mit einfacher als auch mit doppelter Genauigkeit werden in BASIC in einem Sonderformat gespeichert. Es ist nicht nur inkompatibel zu den Formaten der meisten anderen Programmiersprachen, sondern auch zu den Formaten der Arithmetik-Coprozessoren 8087 und 80287.
Damit Sie bei der Programmierung nicht völlig hilflos sind, werden nachfolgend die Schlüsselelemente der BASIC-Fließkommaformate beschrieben. Bei der Darstellung wird davon ausgegangen, daß Sie wissen, wie Fließkommazahlen generell gespeichert und verwaltet werden.
In BASIC unterscheiden sich das einfach- und das doppeltgenaue Datenformat nur in der Anzahl der Mantissenstellen. Einfachgenaue Zahlen belegen vier Bytes, doppeltgenaue acht Bytes. Die Mantisse wird in den ersten drei bzw. sieben Bytes gespeichert, die niederwertigen Bytes zuerst. Der Exponent steht im letzten Byte. Man kann sich das so vorstellen:

M7 M6 M5 M4 M3 M2 M1 E

Wertigkeit: niedrigst höchst

Der Exponent (E) wird als Potenz zur Basis 2 gespeichert, und zwar um 128 nach oben verschoben. Das bedeutet, daß ein Exponent von 0 als 128 (hex 80; 0 + 128 ergibt 128) gespeichert wird, ein Exponent von -3 als 125 (hex 7D; -3 + 128 ergibt 125). Die auf den ersten Blick vielleicht eigenartig erscheinende Verschiebung um 128 wurde gewählt, um in einem einzigen Byte Zahlen von -127 bis +128 abspeichern zu können, ohne daß eine besondere Vorzeichenlogik notwendig ist.
Die Mantisse wird als normale Binärzahl gespeichert. Das höchstwertige Bit des höchstwertigen Bytes (Byte M1) beinhaltet das Vorzeichen. Es ist 0 für positive und 1 für negative Werte. Die höchstwertigen Bits der anderen Bytes dienen der normalen Mantissendarstellung.
Aus dem folgenden Programm können Sie ersehen, wie Fließkommazahlen dekodiert werden. Das Beispiel bezieht sich auf den obigen Text und wird Ihnen helfen, eventuelle Verständnisschwierigkeiten zu beseitigen.

```
100 INPUT "Beliebigen Wert eingeben", EINFACH
110 ADRESSE = VARPTR (EINFACH)
120 PRINT "Die Hex-Bytes sind"
130 FOR I = 0 TO 3
140   H = PEEK (ADRESSE + I)
150   IF H < 16 THEN PRINT "0";
160   PRINT HEX$ (H);" ";
170 NEXT
```

```
180 PRINT
190 E#  = PEEK (ADRESSE + 3)
200 M1# = PEEK (ADRESSE + 2)
210 M2# = PEEK (ADRESSE + 1)
220 M3# = PEEK (ADRESSE + 0)
230 EXPONENT# = E# - 128
240 VORZEICHEN = M1# / 128
250 M1# = 128 + M1# MOD 128
260 MANTISSE# = MI# / 256 + M2# / (256 * 256) + M3# / (256 * 256 * 256)
270 WERT# = MANTISSE# * 2 ^ EXPONENT#
280 IF VORZEICHEN THEN WERT# = - WERT#
290 PRINT "Der dekodierte Wert ist "; WERT#
300 PRINT
320 GOTO 100
```

Beachten Sie, daß die Zeilen 190 bis 220 dazu dienen, die vier Bytes mit dem einfachgenauen Format zu isolieren. Es werden dazu Variablennamen verwendet, die zur Notation der Mantissen-Bytes passen. Wie erwähnt, wird der Exponent in einem um 128 verschobenen Format gespeichert. Zeile 230 isoliert den reinen Exponenten, der von -128 bis +127 reichen kann. Die Zeilen 240 und 250 behandeln das höchstwertige Bit im höchstwertigen Mantissen-Byte, wobei Zeile 240 das Bit als Vorzeichen erkennt und Zeile 250 das Bit für den Wert korrigiert. Anschließend faßt Zeile 260 die drei Mantissen-Bytes zu einem Wert mit dem Dezimalzeichen vor dem ersten Bit zusammen. Zeile 270 fügt den Exponenten hinzu, Zeile 280 sorgt für das richtige Vorzeichen und - Voila! - schon haben wir den dekodierten Wert des BASIC-Fließkommaformates.

20.3.1.3 Stringdatenformate in interpretierendem BASIC

Strings werden in zwei Teilen gespeichert. Der eine Teil hat eine beschreibende Funktion (das ist der *String-Deskriptor*) und enthält Länge und Offsetposition des eigentlichen Strings. Der andere Teil ist der Inhalt des Strings, der aus der ASCII-Zeichenkette besteht.
Der beschreibende Teil hat eine Länge von drei Bytes. Das erste Byte enthält die Länge des Strings, die dadurch auf 255 Zeichen begrenzt ist. Die anderen zwei Bytes stellen den Datensegment-Offset des Stringinhaltes dar. Dieser eigentliche String besitzt kein spezielles Format, er wird als Byte-Folge ab der spezifizierten Stelle gespeichert.
Wird die VARPTR-Funktion auf einen String angewendet, erhält man die Speicherposition des beschreibenden Teils. Aus diesem läßt sich die Offsetadresse des Strings ersehen. Im folgenden Programm wird ein String über diesen Weg gesucht.

```
100 INPUT "Beliebigen String eingeben",STRING$
110 DESKRIPTOR.ADRESSE = VARPTR (STRING$)
120 PRINT "Der Stringdeskriptor liegt ab Offset hex ";
130 PRINT HEX$ (DESKRIPTOR.ADRESSE)
140 STRING.LAENGE = PEEK (DESKRIPTOR.ADRESSE)
150 PRINT "Die Länge des String beträgt ";
160 PRINT STRING.LAENGE
170 STRING.ADRESSE = PEEK (DESKRIPTOR.ADRESSE + 1)
    + 256 * PEEK (DESKRIPTOR.ADRESSE + 2)
180 PRINT "Der Stringinhalt beginnt bei Adresse (hex) ";
190 PRINT HEX$ (STRING.ADRESSE)
200 PRINT "Der Stringinhalt ist ";
210 FOR I = 0 TO STRING.LAENGE - 1
220   PRINT CHR$ (PEEK (I + STRING.ADRESSE));
230 NEXT I
240 PRINT : PRINT
250 GOTO 100
```

20.3.1.4 Stringdatenformate in kompiliertem BASIC

Das Stringformat in kompiliertem BASIC unterscheidet sich von dem eben beschriebenen bezüglich der Abspeicherung der Stringlänge. In interpretierendem BASIC ist die Länge als vorzeichenlose 1-byte-Ganzzahl abgelegt, wodurch sich eine maximale Stringlänge von 255 Zeichen ergibt. In kompiliertem BASIC wird die Stringlänge als vorzeichenbehaftete 2-byte-Ganzzahl gespeichert, ein String kann daher bis zu 32.767 Zeichen umfassen (die negativen Werte bleiben unberücksichtigt).
Auch in kompiliertem BASIC erfolgt der Zugriff auf einen String über einen String-Deskriptor. Die Adresse, die man mit VARPTR(NAME$) erhält, zeigt auf den beschreibenden Teil des Strings, nicht auf den Stringinhalt. Der Deskriptor besteht aus zwei Feldern, die die Länge und den Datensegment-Offset der Zeichenkette enthalten. Der einzige Unterschied beim Stringformat zwischen den beiden BASIC-Versionen ist also, daß interpretierendes BASIC ein Stringlängenfeld von einem Byte verwendet, wohingegen kompiliertes BASIC mit zwei Bytes arbeitet.
Im folgenden Programm wird der Stringzugriff mit VARPTR in kompiliertem BASIC gezeigt. Vergleichen Sie es mit dem Beispiel für interpretierendes BASIC im letzten Abschnitt. Sie werden feststellen, daß nur die Zeilen 140 (Erkennen der Stringlänge) und 170 (PEEK-Offset zur Stringposition) verschieden sind.

```
100 INPUT "Beliebigen String eingeben",STRING$
110 DESKRIPTOR.ADRESSE = VARPTR (STRING$)
120 PRINT "Der Stringdeskriptor liegt ab Offset hex ";
130 PRINT HEX$ (DESKRIPTOR.ADRESSE)
140 STRING.LAENGE = PEEK (DESKRIPTOR.ADRESSE) + 256 * PEEK (DESKRIPTOR.ADRESSE)
150 PRINT "Die Länge des String beträgt ";
160 PRINT STRING.LAENGE
170 STRING.ADRESSE = PEEK (DESKRIPTOR.ADRESSE + 2)
    + 256 * PEEK (DESKRIPTOR.ADRESSE + 3)
180 PRINT "Der Stringinhalt beginnt bei Adresse (hex) ";
190 PRINT HEX$ (STRING.ADRESSE)
200 PRINT "Der Stringinhalt ist ";
210 FOR I = 0 TO STRING.LAENGE - 1
220   PRINT CHR$ (PEEK (I + STRING.ADRESSE));
230 NEXT I
240 PRINT : PRINT
250 GOTO 100
```

20.3.2 Assemblerschnittstellen von interpretierendem BASIC

Wenn Sie in interpretierendem BASIC arbeiten, sollten Sie sich bei Schnittstellenroutinen stets an folgende Konventionen halten:
Alle Parameter werden übergeben, indem ihre Adressen in den Stapel gegeben werden. Man spricht von *Aufruf durch Namen*. Der Vorteil dieser Festlegung ist, daß von der Unterroutine aus die Werte der Parameter so geändert werden können, daß die Änderungen auch für das Hauptprogramm Gültigkeit haben. Der Nachteil liegt in dem komplizierteren Zugriff auf die Parameterwerte, da im Stapel nur die Adressen, aber nicht die Werte stehen.
Parameter werden wie in den meisten anderen Programmiersprachen auch (außer C, siehe Kapitel 20.4.6) in *der* Reihenfolge übergeben, in der sie geschrieben werden. Das bedeutet, daß der erste Parameter am unteren und der letzten am oberen Stapelende zu finden ist. Vorsicht: "Oben" und "unten" eines Stapels sind nicht mit höheren und niederen Adressen gleichzusetzen.
Assemblerroutinen werden mit FAR CALL aufgerufen und mit FAR RET beendet.
Die Assemblerroutinen sind für das Entfernen der Parameter aus dem Stapel verantwortlich. Das ist auch in den meisten anderen Programmiersprachen der Fall (auch hier bildet C eine Ausnahme, lesen Sie in Kapitel 20.5 nach). Das bedeutet unter anderem, daß die Anzahl der Parameter von vornherein feststehen muß.
Das AX-Register wird nicht zur Rückgabe von Werten oder Fehlercodes verwendet. Alle Daten von der Assemblerroutine werden über die Parameter im Stapel übergeben.

Die Segment-, Code- und Stapelregister werden ausschließlich von BASIC verwaltet; alle anderen Register können beliebig verändert werden.
Sehen wir uns nun eine Assemblerschnittstellenroutine an, die auf der Basis des Fünfebenenmodells (siehe Kapitel 8.4.1 ff.) aufgebaut ist. Die Ebenen 1 und 2 können wie folgt aussehen:

```
MEIN_SEG      SEGMENT
              ASSUME   CS:MEIN_SEG
MEINE_PROC    PROC     FAR
                ;die Ebenen 3 bis 5 stehen hier
MEINE_PROC    ENDP
MEIN_SEG      ENDS
END
```

Die Namen MEIN_SEG und MEINE_PROC sind willkürlich gewählt. In interpretierendem BASIC können Sie jeden beliebigen Namen wählen, in kompiliertem BASIC müssen Sie einige Einschränkungen beachten.
Wenn Sie die Routine mit der allgemeinen Schnittstellenroutine in Kapitel 8.4.1 vergleichen, werden Ihnen zwei Unterschiede auffallen. Im vorliegenden Beispiel folgt auf die Anweisung SEGMENT nicht die Klassifikation 'CODE' und es kommt keine Anweisung PUBLIC MEINE_PROC vor. Die Abweichungen resultieren aus der Tatsache, daß der Name der Prozedur für BASIC bedeutungslos ist, da ein Programm in interpretierendem BASIC nicht lauffähig gemacht werden muß. Eine Assemblerschnittstellenroutine braucht also nicht mit einem Programm in interpretierendem BASIC gebunden werden. Bei Programmen, die von DOS-LINK bearbeitet werden, sind die Schlüsselworte CODE und PUBLIC von größter Bedeutung. Das trifft auf alle anderen Programmiersprachen in diesem Kapitel zu.
Die Ebene 3 umfaßt die Basiszeigersicherung und die Vorbereitungen zur Parameterübergabe. Für interpretierendes BASIC sieht das folgendermaßen aus:

```
PUSH  BP
MOV   BP,SP
                ;Ebene 4 und 5 stehen hier
POP   BP
RET   xxx
```

Die vier Instruktionen sind standardisiert und sollten für alle Assemblerschnittstellenroutinen verwendet werden. xxx steht stellvertretend für die Anzahl der Parameter-Bytes, die mit POP aus dem Stapel geholt werden sollen. Der Wert von xxx muß doppelt so groß sein wie die Anzahl der Parameter, die beim Aufruf durch CALL an die Routine übergeben werden. Sind z.B. drei Parameter zu Beginn transferiert worden, sollte xxx gleich 6 sein, damit der Stapel gesäubert werden kann. Werden keine Parameter

übergeben, nimmt man den Wert 0. Nebenbei bemerkt wird der Befehl RET beim Assemblieren zu FAR RET, da am Anfang der Routine die Anweisung PROC FAR steht.
Auf der nächsten Ebene, 4, wird der Parameterzugriff festgelegt. Wie Sie vielleicht schon dem Kapitel 8.4.5 entnommen haben, steht die Adresse des letzten Parameters im Stapel an der Speicherstelle [BP+6], die des vorherigen Parameters bei [BP+8] usw.
Um Ihnen auch hier ein Beispiel zu geben, lassen Sie uns annehmen, Sie wollen drei Ganzzahlparameter in die Register AX, BX und CX laden (aus welchem Grund auch immer). Hier der Code der Ebene 4, der diese Aufgabe erfüllt:

```
MOV   SI,[BP+10]   ;Adresse des ersten Parameters holen
MOV   AX,[SI]      ;ersten Wert holen
MOV   SI,[BP+8]    ;Adresse des zweiten Parameters holen
MOV   BX,[SI]      ;zweiten Wert holen
MOV   SI,[BP+6]    ;Adresse des dritten Paramters holen
MOV   CX,[SI]      ;dritten Wert holen
```

Bachten Sie, daß die Parameterwerte in zwei Schritten geholt werden: Zuerst wird die Adresse festgestellt (und im SI-Register abgelegt, weil das ein günstiger Platz ist), dann erfolgt der Zugriff auf den Parameterwert über die Adresse. Der Wert kann in jede beliebige freie Stelle übernommen werden, im Beispiel sind es die Register AX, BX und CX.
Damit ist aber nur die erste Hälfte der Ebene 4 abgehandelt, die Parameterübergabe vom BASIC- zum Assemblerprogramm. Im zweiten Teil geht es um den umgekehrten Vorgang. Sie erinnern sich: Der Datenaustausch von der Assemblerroutine zum Hauptprogramm erfolgt ebenfalls über die Parameter. Angenommen, in der Routine sei ein Wert berechnet und in Register DX abgelegt worden. Der Wert soll an das BASIC-Programm als erster Parameter übergeben werden. Die Anweisungen dazu lauten:

```
MOV  SI,[BP+10]    ;Adresse des ersten Parameters (erneut) holen
MOV  [SI],DX       ;den in DX stehenden Wert zurückgeben
```

Zum Abschluß sehen wir uns eine komplette Assemblerroutine an, die die Werte der Register SS und SP an das Hauptprogramm übergibt. Damit ist es möglich, daß ein BASIC-Programm Informationen über den eigenen Stapel erhält. Es gibt kaum einen Grund, eine solche Routine einzusetzen, zum Experimentieren ist sie aber gut geeignet. Die Routine muß mit zwei Ganzzahlparametern aufgerufen werden. Der Wert der Parameter ist ohne Belang, sie dienen nur dazu, einen Platz für die SS- und SP-Inhalte bereitzustellen. Hier das Beispiel:

```
STAPELINFO SEGMENT
           ASSUME  CS:STAPELINFO
STAPEL     PROC    FAR
           PUSH    BP
           MOV     BP,SP
           MOV     SI,[BP+8]
           MOV     [SI],SS
           MOV     SI,[BP+6]
           MOV     [SI],SP
           POP     BP
           RET     4
STAPEL     ENDP
STAPELINFO ENDS
           END
```

20.3.3 Assemblerschnittstellen in kompiliertem BASIC

Jeder, der sich ausführlich mit dem BASIC der PC-Modelle beschäftigt hat, kann sein Leid klagen, wenn es um die teilweise verrückten Inkompatibilitäten von interpretierendem und kompiliertem BASIC geht. Wir werden hier nur die wichtigsten Unterschiede, die für Assemblerschnittstellen relevant sind, herausstellen und ein Beispiel durcharbeiten.
In interpretierendem BASIC ist es üblich, Schnittstellenroutinen an einer freien Stelle im Speicher abzulegen und eine Variable anzulegen, die auf die Routine verweist. Aufgerufen wird die Routine mit CALL und dem Variablennamen. Diese Methode ist fehlerträchtig und nicht sehr elegant.
In kompiliertem BASIC gibt es zwei Möglichkeiten zur Integration einer Assemblerroutine. Die erste entspricht weitgehend der für interpretierendes BASIC beschriebenen Methode, hat jedoch eine etwas andere Form. Beispiel: Es soll POSITION% als Variablenname der ersten Speicherstelle einer Assemblerschnittstelle und PARAMETER für alle zu übergebenden Parameter stehen. In interpretierendem BASIC würde man die Schnittstelle folgendermaßen aufrufen:

```
CALL POSITION% (PARAMETER)
```

In kompiliertem BASIC sieht der Aufruf anders aus, das Endergebnis ist allerdings das gleiche:

```
CALL ABSOLUTE (PARAMETER,POSITION%)
```

In kompiliertem BASIC muß POSITION% eine Ganzzahlvariable sein, in interpretierendem BASIC ist jedes Variablenformat erlaubt.
Die Anweisung CALL ABSOLUT ist einer näheren Betrachtung wert. In kompiliertem BASIC sind *alle* Aufrufe in Wirklichkeit normale Aufrufe von externen Routinen, die an die kompilierten Programme angebunden

werden. Das ist in fast allen Compiler-Sprachen Standard. Das Äquivalent zu CALL POSITION% in interpretierendem BASIC simuliert kompiliertes BASIC nun mit Hilfe der Bibliotheks-Unterroutine ABSOLUTE. In anderen Worten, ABSOLUTE ist kein Teil von BASIC selbst in dem Sinne wie ON ERROR oder CHAIN Teile von BASIC sind, sondern ABSOLUTE ist der Name einer externen Routine, die in der LINK-Bibliothek von BASIC zu finden ist. Die Routine verwendet die Parameter (einschließlich des Parameters POSITION%), um die Operation CALL POSITION%, die in interpretierendem BASIC enthalten ist, zu simulieren.
Die Vorgehensweise bei der Ablage einer Schnittstellenroutine im Speicher ist bei interpretierendem und kompiliertem BASIC gleich. Entweder wird die BLOAD-Anweisung verwendet, um die Schnittstelle aus einer Datei zu laden oder die Schnittstelle wird Byte für Byte mit dem Befehl POKE in den Speicher gebracht. Die schwierige Frage ist nicht, *wie* die Routine abgelegt, sondern *wo* sie im Speicher plaziert werden kann. Die Antwort hängt von so vielen Umständen ab, daß wir das Problem in diesem Buch ausklammern müssen. Das Thema ist eher als Gegenstand eines Buches über fortgeschrittene BASIC-Programmierung geeignet.
In kompiliertem BASIC existiert eine Alternative zu der Methode, eine Assemblerroutine "einfach so" im Speicher abzulegen. Man kann mit LINK arbeiten. Module wie Assemblerschnittstellen werden separat vorbereitet und entweder in Form einer einzelnen Objektdatei (mit der Erweiterung .OBJ) oder innerhalb einer Bibliothek (mit der Erweiterung .LIB) abgespeichert. Die Verknüpfung von BASIC-Programm und Assemblerschnittstelle geschieht mit LINK, wie in Kapitel 19 erklärt.
Gleichgültig, mit welcher Methode die Schnittstellenroutine installiert wird, stellt sich die Frage, wie die Routine anzusprechen ist. In BASIC wird die Schnittstelle mit

```
CALL NAME (PARAMETER)
```

augerufen. PARAMETER hat dieselbe Bedeutung wie für die zuvor beschriebene Methode (oder in interpretierendem BASIC) erläutert. NAME ist der Name, der in der Assemblerschnittstelle erscheint und die gewünschte Routine wählt.
Bezüglich der Stapelbenutzung, Parameterübergabe und der Aufrufsart (FAR CALL erfordert FAR RET) bestehen zwischen interpretierendem und kompiliertem BASIC aus der Sicht des Programmierers keine Unterschiede. Daher wird im folgenden das bereits vom interpretierenden BASIC her bekannte Beispiel herangezogen und es werden nur die Unterschiede herausgestellt.

```
STAPELINFO SEGMENT
          ASSUME  CS:STAPELINFO
          PUBLIC  STAPEL
```

```
STAPEL      PROC    FAR
            PUSH    BP
            MOV     BP,SP
            MOV     SI,[BP+8]
            MOV     [SI],SS
            MOV     SI,[BP+6]
            MOV     [SI],SP
            POP     BP
            RET     4
STAPEL      ENDP
STAPELINFO  ENDS
            END
```

Die einzige Veränderung, die für kompiliertes BASIC nötig ist, besteht darin, daß der Name der Routine als PUBLIC und damit als offizieller Name, den der Assembler verwenden kann, deklariert werden muß. Das ist nötig, damit die Routine an ein kompiliertes Programm angebunden werden kann. Der Befehl dazu ist PUBLIC STAPEL in der dritte Zeile. Es ist der einzige Unterschied zwischen dieser Routine und der im letzten Abschnitt vorgestellten. Die PUBLIC-Anweisung darf auch in der Routine für interpretierendes BASIC vorkommen, bleibt dort jedoch ohne Wirkung.
Nachdem Sie die Assemblerseite kennengelernt haben, wollen wir uns nun die BASIC-Seite ansehen. Auch dazu ein Beispielprogramm:

```
100 'Beispielprogramm für den Aufruf der Routine STAPEL
110 '
120 I% = 0
130 J% = 0
140 CALL STAPEL(I%,J%)
150 PRINT "In der Mitte eines Programms"
160 PRINT " ist das Stapelsegment ";HEX$(I%)
170 PRINT " ist der Stapelzeiger ";HEX$(J%)
180 GOSUB 200
190 GOTO 270
200 'Unterroutine
210 '
220 CALL STAPEL(I%,J%)
235 PRINT "In der Unterroutine "
240 PRINT " ist das Stapelsegment ";HEX$(I%)
250 PRINT " ist der Stapelzeiger ";HEX$(J%)
260 RETURN
270 END
```

Das Programm zeigt einige bemerkenswerte Einzelheiten. Eine Assemblerroutine wird mit dem Namen (STAPEL), der in der PUBLIC-Anweisung im Assemblercode steht, aufgerufen (Zeilen 140 und 220). Die Namen im

BASIC- und im Assemblerprogramm müssen übereinstimmen, damit der LINK-Vorgang erfolgreich verläuft. Der Name des Segmentes im Assemblerprogramm, STAPELINFO, ist willkürlich gewählt und ohne Bedeutung. Die Anzahl der verwendeten Parameter soll bei aufrufendem und aufgerufenem Programm identisch sein. Die Anweisung RET 4 am Ende der Assemblerroutine nimmt vier Parameter-Bytes aus dem Stapel, was zwei Parametern zu je zwei Bytes entspricht.

Hinweis: Mit Tricks kann man erreichen, daß die Anzahl der Parameter variabel sein darf. Darauf wollen wir hier nicht näher eingehen.

Unser Beispielprogramm meldet den Status des Stapels in zwei verschiedenen Situationen: einmal im linearen Programmablauf und zum anderen in einem Unterprogramm. Der Unterschied zwischen den beiden Stapelzeigerwerten gibt Einblick in die Stapelbenutzung des Compilers.
Wir haben eine Assemblerroutine und ein BASIC-Programm erstellt. Nun wird es Zeit, beide miteinander zu verknüpfen. Wie Sie wissen, geschieht das mit LINK. Nehmen wir an, der Assemblerquellcode befinde sich in der Datei STAPEL.ASM. Mit dem Kommando

```
MASM STAPEL;
```

erreichen wir, daß das Programm in die Datei STAPEL.OBJ assembliert wird. Nehmen wir weiter an, das BASIC-Programm stehe in der Datei TEST.BAS. Es muß sich um eine ASCII-Datei handeln, das heißt, die BASIC-Befehlsworte dürfen nicht in Tokens umgesetzt sein. Um das Programm zu kompilieren, geben Sie ein:

```
BASCOM TEST;
```

Das Ergebnis ist eine Datei TEST.OBJ, die den Objektcode enthält. Nun gilt es, die beiden Objektdateien zu verbinden:

```
LINK TEST+STAPEL
```

Die endgültige Programmdatei enthält das (kompilierte) BASIC-Programm mit der Assemblerroutine und trägt den Namen TEST.EXE.

20.4 Pascal

In diesem Abschnitt werden wir den IBM PC Pascal Compiler und den ähnlich funktionierenden Microsoft Pascal Compiler besprechen. Bei den verschiedenen Datenformate wird erwähnt, welche Aspekte für den einen oder den anderen Compiler gelten und welche Formate Standard in Pascal sind. Falls Sie mit einem anderen Compiler als den beiden genannten arbeiten, sollten Sie die nachfolgenden Erläuterungen dennoch durchlesen,

um sich ein Grundlagenwissen über Pascal-Assembler-Schnittstellen anzueignen. Mit dem Compiler-Handbuch sollte die Anpassung an Ihr System nicht schwer fallen.
Beachten Sie, daß zwischen den Versionen 1 und 2 des IBM Pascal Compilers enorme Unterschiede existieren. Im Text wird an den entsprechenden Stellen darauf hingewiesen.

20.4.1 Pascal-Datenformate

Es gibt drei geläufige Datenformate in Pascal: Integer (Ganzzahl), Fließkomma (in der Pascal-Terminologie auch als *Real* bekannt) und String. Außerdem existiert noch das Format *Set*. Innerhalb der Datenformate gibt es einige unterschiedliche Datentypen, vor allem bei Ganzzahlformaten.

20.4.1.1 Ganzzahldatenformate

Werte im Ganzzahl- oder Integerdatenformat werden als Binärzahlen mit dem niederwertigem Byte an erster Stelle im Speicher abgelegt. Integerzahlen können eine Länge von einem, zwei oder vier Bytes besitzen und mit oder ohne Vorzeichen ausgestattet sein. Das ergibt sechs vorstellbare Formate, von denen fünf gängig sind, das sechste (vier Bytes ohne Vorzeichen) wird nicht verwendet. Einige der Formate erfüllen aus der Sicht des Pascal-Programmierers gleiche Aufgaben.
Beginnen wir mit 1-byte-Integerzahlen. Das Format mit Vorzeichen heißt SINT (*Short Integer*) und weist einen Wertebereich von 128 bis +127 auf. Die vorzeichenlose Form BYTE verfügt über einen Bereich von 0 bis 255. Weder SINT noch BYTE sind Teil von Standard-Pascal. Da sie in der Compiler-Dokumentation nur kurz beschrieben werden, übersieht man sie leicht.
Auch das 2-byte-Integerformat gibt es mit und ohne Vorzeichen. Im ersten Fall erstreckt sich der Wertebereich von 32.768 bis +32.767; das Format heißt INTEGER. Die vorzeichenlose Form wird WORD genannt und kann Zahlen von 0 bis 65.535 aufnehmen. INTEGER ist ein Pascal-Standardformat, WORD nicht. Das INTEGER-Format entspricht dem 16-bit-Ganzzahlformat von BASIC, es ist in fast allen Programmiersprachen anzutreffen. Das vorzeichenlose 16-bit-Format WORD hat in der Sprache Pascal eine eigene Bedeutung. Es wird für alle Adressierungen wie ADR oder ADS verwendet (weitere Hinweise zur Adressierung in Pascal finden Sie in Kapitel 20.4.2). Das WORD-Format entspricht dem standardisierten vorzeichenlosen 16-bit-Ganzzahlformat, wie es z.B. als UNSIGNED INT in C verwendet wird.

4-byte-Integerzahlen wurden mit der Compiler-Version 2.0 eingeführt, sie tragen immer ein Vorzeichen. Der Wertebereich liegt zwischen 2.147.483.648 und +2.147.483.647, der Name ist INTEGER4. Dieses 32-bit-Format ist weder in Pascal noch in anderen Programmiersprachen ein Standard, in vielen ist es überhaupt nicht vorhanden. C stellt ein identisches Format als LONG INT zur Verfügung. INTEGER4 ist nicht voll in die Ganzzahlformate von Pascal integriert, in vielen Anweisungen muß eines der anderen Integerformate gewählt werden.
In Standard-Pascal steht ein universelles Ganzzahldatenformat, der sog. *Aufzählungstyp*, zur Verfügung. Für praktisch alle Anwenddungen wird das Aufzählungsformat zur Ablage positiver ganzer Zahlen ab 0 benutzt, wobei den einzelnen Werten Namen zugeordnet sind. Intern besteht das Aufzählungsformat aus vorzeichenlosen Integerzahlen von ein (BYTE) oder zwei (WORD) Bytes Länge, abhängig davon, ob die Anzahl der Werte in ein Byte paßt oder nicht. Der Datentyp BOOLEAN, eines der am häufigsten verwendeten Spezialdatenformate, ist nichts anderes als ein vordefinierter Aufzählungstyp mit zwei Werten. BOOLEAN belegt nur ein Byte und kann die Werte 0 und 1 annehmen.

20.4.1.2 Stringdatenformate

Standard-Pascal verfügt über das Datenformat CHAR, das man je nach Denkweise als einen Spezialfall des BYTE-Formates oder des STRING-Formates (oder LSTRING) ansehen kann, wenngleich CHAR in Pascal ein eigenständiges Format ist. CHAR speichert ein einzelnes ASCII-Zeichen in einem Byte.
Es gibt in Pascal zwei Stringformate: STRING gehört zu Standard-Pascal und besteht aus einer festgelegten Anzahl von ASCII-Zeichen, LSTRING ist eine Erweiterung zum Standard-Pascal und kann Strings variabler Länge enthalten. Fast alle Compiler stellen die Möglichkeit zum Umgang mit Strings variabler Länge zur Verfügung, jedoch leider auf zum Teil sehr unterschiedliche Weisen. Wir besprechen das Format der IBM/Microsoft Compiler.
Strings fester Länge werden als eine Byte-Kette ohne besondere Formatierungen gespeichert. Zur Notation in Pascal: Ist S ein String fester Länge, wird das erste Zeichen des String mit S[1] bezeichnet. Die Adresse eines String mit fester Länge ist zugleich die Adresse des ersten Zeichens des Strings.
Bei einem String variabler Länge steht vor der Byte-Zeichenkette ein Byte, das die Stringlänge als vorzeichenlose Ganzzahl angibt. Daraus folgt, daß LSTRINGS aus maximal 255 Zeichen bestehen können. Das erste Element der Zeichenkette ist S[1], genau wie bei Strings mit fester Länge. S[0] bezeichnet das Byte, das die Stringlänge enthält. Die Adresse eines Strings mit variabler Länge ist nicht die des ersten Zeichens, sondern der Längenangabe.

Beachten Sie, daß in Pascal die Länge eines String explizit festgelegt werden muß. Bei Strings fester Länge ist das die Anzahl der Zeichen, bei Strings variabler Länge kommt ein Byte mehr hinzu.

20.4.1.3 SET-Datenformat

Das SET-Datenformat ist eine Besonderheit von Pascal. Die wenigsten anderen Programmiersprachen verfügen über einen vergleichbaren Datentyp. Ein sog. *Set* basiert auf dem Aufzählungsdatentyp mit nicht mehr als 256 Elementen. Im SET-Datenformat wird jedem Element des zugrundeliegenden Aufzählungstyps ein Bit zugeordnet, das gesetzt wird, wenn das Element zum Set gehört. Die Länge des SET-Formates ist von der Anzahl der Elemente im Aufzählungsformat abhängig, da für jedes Element ein Bit benötigt wird. Die kleinste SET-Einheit besteht allerdings aus zwei Bytes, so daß Sets aus zwei, vier, sechs etc. bis maximal 32 Bytes gebildet werden können. Ein Set mit zwei Bytes kann bis zu 16 Elemente umfassen, ein Set mit 32 Bytes bis zu 256 Elemente. Im Gegensatz zu dem, was Sie vielleicht erwarten, werden Set-Daten in 2-Byte-Einheiten abgespeichert. Ein Set für acht Elemente nimmt also eine Speicherkapazität von zwei Bytes ein und nicht, wie zu vermuten wäre, ein Byte, in dem es theoretisch untergebracht werden könnte. Die minimale Länge eines Set beträgt also zwei Bytes (für Sets von 0 bis 16 Elementen), die maximale Länge beträgt 32 Bytes (für Sets, die auf einem Aufzählungstyp zwischen 241 und 256 Elementen aufgebaut sind).
Die Bits werden von links nach rechts zugeordnet und die Kodierung erfolgt in der Reihenfolge der Deklaration der Elemente im Aufzählungsdatentyp.

20.4.1.4 Fließkommadatenformate

Fließkommadatenformate, in Pascal spricht man von *Real-Zahlen*, stellen uns vor einige interessante Probleme. Es gibt drei Formate, die erwähnenswert sind. Das erste Format existiert nur in der Compiler-Version 1.0 und entspricht dem BASIC-Format für einfachgenaue Zahlen. Wir wollen es im folgenden *Früh-Real* nennen, was nur der Unterscheidung dient und keine offizielle oder übliche Bezeichnung darstellt. Die anderen beiden Fließkommadatenformate beziehen sich auf Compiler-Auslieferungen ab Version 2 und entsprechen dem Standardformat des 8087 Arithmetik-Coprozessors. In Pascal heißen die beiden neueren Fließkommaformate REAL4 und REAL8, wir wollen sie in diesem Kapitel als *Spät-Real* zusammenfassen. Eine praktikable Umwandlungsmöglichkeit von *Früh-Real* in *Spät-Real* gibt es nicht, wenngleich die Konvertierung grundsätzlich möglich ist.

Beginnen wir mit dem Format *Früh-Real*. Es entspricht dem BASIC-Format für einfachgenaue Zahlen und wird unter diesem Aspekt in Kapitel 20.3.1 beschrieben. In der folgenden Darstellung wird davon ausgegangen, daß sie mit den Grundlagen der Fließkommaarithmetik und der Speicherung von Fließkommazahlen vertraut sind. Es werden lediglich die Besonderheiten des Formates *Früh-Real* erläutert.
Das Format Früh-Real wird in vier Bytes gespeichert, die wie folgt zusammengefaßt werden können:

M3 M2 M1 E

Die Bytes M1 bis M3 stellen die Mantissen-Bytes dar, das höchstwertige Bit in M1 repräsentiert das Vorzeichen des Wertes (1 bedeutet negatives Vorzeichen). Das E-Byte enthält den binären Exponenten in einer um 128 nach oben verschobenen Form. Das bedeutet, um den wahren Exponenten zu erhalten, muß von dem in E abgelegten Wert 128 subtrahiert werden. Die Mantisse stellt den Nachkommateil der Zahl dar, das heißt, das (gedachte) Dezimalzeichen liegt links des höchswertigen Bits.
Die Formate Spät-Real unterscheiden sich in allen wesentlichen Punkten vom alten Format, sind untereinander aber praktisch gleich; lediglich die Länge differiert.
Jedes der beiden neuen Formate besteht aus drei Feldern, einem Vorzeichen, einem Exponenten und einer Mantisse, die in dieser Reihenfolge mit den niederwertigen Bits zuvorderst gespeichert werden. Mit anderen Worten, das höchstwertige Bit des "rechten" Bytes stellt das Vorzeichen dar, das zweithöchste Bit dieses Bytes das höchswertige Bit des Exponenten usw. Die niederwertigen Bits der Mantisse stehen also im ersten ("linken") Byte.
Das Vorzeichenfeld beider Formate besteht aus einem Bit, 1 bedeutet negatives, 0 positives Vorzeichen.
Der Exponent wird als im Bereich verschobene ganze Zahl gespeichert. Bei REAL4 besteht das Exponentenfeld aus acht Bits, der Wert ist um 127 nach oben verschoben. Das bedeutet, wenn Sie 127 subtrahieren, ergibt sich der wahre Exponent. Im REAL8-Format ist das Feld elf Bits groß und speichert den Exponenten um 1023 verschoben. Der Wertebereich des Exponenten erstreckt sich bei REAL4 von 127 bis +128, bei REAL8 von 1023 bis +1024. Zur Erinnerung: Das Format *Früh-Real* (und das entsprechende BASIC-Format) arbeitet mit Exponenten zwischen -128 und +127. Beachten sie außerdem, daß sich REAL4 und REAL8 im Gegensatz zu den BASIC-Formaten *einfach-* und *doppeltgenau* nicht nur in der Mantissengenauigkeit, sondern auch im Exponentenwertebereich unterscheiden.
Das Mantissenfeld enthält den absoluten Dezimalteil der Zahl, das Vorzeichen wird - wie dargelegt - getrennt im höchstwertigen Bit abgelegt. Das entspricht dem Format *Früh-Real*. REAL4 verfügt über 23 Mantissen-Bits (ohne Vorzeichen), REAL8 über 52 Mantissen-Bits (ohne Vorzeichen).

REAL4 belegt insgesamt 32 Bits oder vier Bytes, REAL 8 beansprucht 64 Bits oder acht Bytes. Beachten Sie, daß sich die drei Felder (Vorzeichen, Exponent, Mantisse) nicht an Byte-Grenzen orientieren. Hier eine schematische Darstellung:

REAL4				REAL8							
M3	M2	E2M1	VE1	M7	M6	M5	M4	M3	M2	E2M1	VE1

Die Mantissen-Bytes M2 bis M7 enthalten ausschließlich Mantissen-Bits. Im Byte VE1 befinden sich neben dem Vorzeichen-Bit die ersten sieben Bits des Exponenten:

Bit 7	6	5	4	3	2	1	0	Beschreibung
V	.	.	.	.	.	.	.	Vorzeichen-Bit
.	E	E	E	E	E	E	E	Höchstwertige Exponenten-Bits

Das Byte E2M1 beinhaltet die restlichen Exponenten-Bits (ein Bit bei REAL4 und vier Bits bei REAL8) und die ersten sieben bzw. vier Bits der Mantisse:

REAL4 Bits								REAL8 Bits								
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	Beschreibung
E	.	.	.	.	.	.	.	E	E	E	E	.	.	.	.	Niederwertigste Exponenten-Bits
.	M	M	M	M	M	M	M	.	.	.	.	M	M	M	M	Mantissen-Bits

Wenn man *Früh-Real* und *Spät-Real* vergleicht, stellt man fest, daß die Formate von der Idee her praktisch gleich sind, sich aber in nahezu allen Details voneinander unterscheiden.
In Kapitel 20.3.1.2 finden Sie ein Programm, daß die Format-Bytes einer BASIC-Fließkommazahl dekodiert und auf dem Bildschirm darstellt. Für die Pascal-Fließkommaformate lassen sich ähnliche Programme schreiben, wobei die Besonderheiten der verschiedenen Formate zu berücksichtigen sind. Machen Sie sich nochmals klar: *Früh-Real* ist kompatibel mit BASIC, *Spät-Real* ist kompatibel mit den meisten anderen Programmiersprachen wie z.B. IBM/Microsoft FORTRAN oder Lattice/Microsoft C und mit dem 8087/80287-Fließkommaformat.

20.4.2 Assemblerschnittstellen in Pascal

In diesem Abschnitt erhalten Sie die wichtigsten Informationen über den Parameteraustausch zwischen Pascal und Assembler. Wir entlehnen zu diesem Zweck das Pascal-spezifische Beispiel aus Kapitel 10.4 und entfernen die nebensächlichen Teile:

```
SCHNITTSTELLE SEGMENT 'CODE'    ;1
              PUBLIC  SEGLESEN  ;2
SEGLESEN      PROC    FAR       ;3
              PUSH    BP        ;4
              POP     BP,SP     ;5
              ...               ;6
              POP     BP        ;7
              RET     10        ;8
SEGLESEN      ENDP              ;9
SCHNITTSTELLE ENDS              ;10
              END               ;11
```

Der Name SCHNITTSTELLE in Zeile 1 ist willkürlich wählbar, die Klassifikation 'CODE' in einfachen Anführungszeichen ist nötig, damit die Routine mit dem Pascal-Compiler gebunden werden kann.
In Zeile 2 wird der Name der Routine, SEGLESEN, als PUBLIC deklariert. Das ist nötig, damit der Binder die Routine mit den Programmen, von denen aus sie aufgerufen wird, zusammenfügen kann. SEGLESEN (PROC) wird in Zeile 3 als FAR-Prozedur definiert, weil Pascal alle externen Routinen mit einem FAR-Aufruf anspricht.
Für die Parameterübergabe über den Stapel gibt es, wie Sie vermutlich wissen, zwei Methoden: Entweder wird der Parameterwert direkt in den Stapel geschrieben, oder aber nur eine Adresse, die auf den Parameter verweist. Jede Methode hat ihre Vorteile. Die direkte Übergabe des Parameterwertes vereinfacht den Zugriff von der aufgerufenen Routine aus, die Übergabe der Adresse erleichtert es, den Parameterwert durch die Routine verändern zu lassen. Nun, in Pascal stehen in gewissen Grenzen beide Wege offen - im Unterschied beispielsweise zu BASIC, wo nur die Adreßübergabe möglich ist.
Bei der Festlegung einer Routine wird ein Parameter als VAR oder VARS definiert, wenn die Parameteradresse im Stapel übergeben werden soll. Im Pascal-Sprachgebrauch bedeutet das, daß einer Routine erlaubt wird, Parameterwerte zu verändern. Eine Ebene tiefer handelt es sich darum, daß Adressen und nicht Parameterwerte in den Stapel gegeben werden. Bei VAR wird der Datensegment-Offset der Adresse als ein 2-byte-Wort übergeben, bei VARS die vollständige segmentierte Adresse mit einem Segmentadressen- und einem relativen Offset-Teil als zwei 2-byte-Worte. Die Unterscheidung zwischen VAR und VARS ist wichtig, weil am Ende der Unterroutine alle Parameter mit RET aus dem Stapel entfernt werden müssen und dazu bekannt sein muß, wieviele Parameter-Bytes vorhanden sind.
Wird VAR nicht spezifiziert, schützt Pascal den Parameter vor einer bleibenden Veränderung durch die Unterroutine. Das bedeutet aber nicht, daß automatisch der Parameterwert in den Stapel geschrieben wird. Vielmehr kann der Schutz auf zwei Arten erfolgen. Nur wenn der Parameterwert in den Stapel paßt (z.B. Ganzzahlwert), wird er in den Stapel über-

geben. Andernfalls (z.B. Zeichenkette) wird eine Sicherheitskopie des Wertes im Speicher angelegt und die Adresse der Kopie in den Stapel gegeben. Die Unterscheidung ist aus zwei Gründen wichtig. Erstens muß die jeweilige Methode beim Parameterzugriff von der Unterroutine aus berücksichtigt werden und zweitens ist die Adressenübergabe für das aufrufende Programm weniger effizient als die direkte Wertübergabe.
Aus den Erklärungen der letzten Abschnitte ergibt sich, daß bereits beim Erstellen der Assemblerroutine bekannt sein muß, welche Parameter in welcher Form zu transferieren sind. Wenn Sie sich nicht klar darüber sind, ob unter gegebenen Umständen Werte oder Adressen im Stapel übergeben werden - testen Sie! Experimentieren Sie mit unterschiedlichen Programmen und sehen Sie sich den vom Compiler erzeugten Code an. Sie werden feststellen, daß sich der Compiler in den meisten Fällen genau so verhält, wie man es erwarten würde.
Sind Sie dennoch unsicher, spezifizieren Sie einfach alle Parameter mit VAR, um damit die Adressenübergabe zu erzwingen. Eine einheitliche Verwendung der VAR-Festlegung hat zudem den großen Vorteil, daß eine gleichartige Behandlung aller Parameter möglich ist. Der einzige Vorteil von VARS gegenüber VAR besteht darin, daß man sich zwei Assembleranweisungen erspart. Dieser Vorteil ist nur als marginal anzusehen.
Von der Unterroutine kann ein Wert an das aufrufende Programm zurückgegeben werden, wenn sie als FUNCTION deklariert wird. Ist der zurückgegebene Wert hinlänglich einfach, erwartet das aufrufende Programm, daß er im AX-Register übergeben wird. Falls das nicht der Fall ist, wird der Wert im Speicher abgelegt und die Offset-Adresse hinter den Parametern der Funktion in den Stapel gegeben. Pascal legt dann einen neuen Parameter des Typs VAR für die Funktionsunterroutine an und der Funktionswert der Unterroutine wird an den Parameter übergeben.
Um zu zeigen, wie Parameter durch den Stapel übergeben werden, lassen Sie uns ein Beispiel durcharbeiten. Nehmen wir an, die Assemblerschnittstellenroutine ist in Pascal wie folgt deklariert:

```
PROCEDURE  BEISPIEL (I : INTEGER; VAR J : INTEGER);
EXTERNAL;
```

Die Assemblerroutine findet den ersten Parameter, I, der im Stapel bei Offset [BP+8] liegt. Um den Wert in Register AX zu übergeben, schreiben wir in Assembler:

```
MOV     AX,[BP+8]
```

Der zweite Parameter, J, steht im Stapel bei Offset [BP+6]. Um den Wert in Register AX zu bringen, benötigen wir zwei Schritte: erst wird die Adresse aus dem Stapel geholt und anschließend über die Adresse auf den Wert zugegriffen.

```
MOV     BX,[BP+6]
MOV     AX,[BX]
```

Um den Wert zu verändern, muß der Prozeß quasi umgekehrt werden:

```
MOV   BX,[BP+6]
MOV   [BX],AX
```

Der Wert von I läßt sich nicht verändern, da nicht die Adresse, sondern nur eine Kopie des Parameterwertes, im Stapel steht.
Mit diesem Grundlagenwissen dürften Sie in der Lage sein, Pascal-Programme mit Assemblerschnittstellen zu versehen. Allerdings werden Sie nicht darum herumkommen, im einen oder anderen Fall ein wenig zu experimentieren, bevor die Schnittstelle reibungslos funktioniert.

20.5 Programmiersprache C

Die nachfolgenden Ausführungen beziehen sich auf den Lattice/Microsoft C-Compiler, Version 1.04, gelten jedoch auch für viele gleichgeartete Compiler. Selbst wenn Sie mit einem völlig anderen C-Compiler arbeiten, sollten Sie den Abschnitt lesen, da die Grundlageninformationen von genereller Bedeutung sind.
Man unterscheidet Compiler unter anderem danach, welche Speicherkapazität sie verwalten können. Beim *kleinen Speichermodell* sind für Programmtext (Codesegment) und Daten gemeinsam 64 Kbyte vorgesehen. Der Lattice/Microsoft C-Compiler arbeitet mit dieser Restriktion.
Das *mittlere Speichermodell* stellt für das Codesegment einen unbegrenzten Speicherbereich zur Verfügung, der Datenbereich ist auf 64 Kbytes beschränkt. Beim *großen Speichermodell* werden sowohl für das Programm als auch für die Daten segmentierte Adressen verwendet, wodurch beide Teile größer als 64 Kbyte sein können. Kompiliertes BASIC und Pascal arbeiten mit dem *mittleren Speichermodell*, obwohl Pascal in begrenztem Maße segmentierte Adressen für Daten verwendet. Bei späteren Versionen des Lattice C-Compilers hat man die Möglichkeit, mit allen drei Speichermodellen zu arbeiten. Darauf wollen wir nicht weiter eingehen, sondern uns nur dem kleinen Speichermodell mit 64 Kbyte für Programm und Daten zusammen widmen. Die damit verbundene Speicherverwaltung ist wesentlich effizienter als die Verwaltungen für größere Kapazitäten.

20.5.1 C-Datenformate

In C gibt es drei Hauptdatenformate: Integer (Ganzzahl), Fließkomma und String. Der Lattice/Microsoft C-Compiler verfügt über vier unterschiedliche Integer- und zwei Fließkommaformate.

20.5.1.1 Ganzzahldatenformate

Wir beginnen mit den vier Integerformaten: CHAR, INT, UNSIGNED und LONG. Das CHAR-Format beansprucht ein Byte mit einer ganzen Zahl zwischen 0 und 255. Das INT-Format ist ein 2-byte-Wort mit Vorzeichen und einem Wertebereich von -32.768 bis +32.767. Das UNSIGNED-Format als 2-byte-Wort ohne Vorzeichen kann Werte zwischen 0 und 65.535 aufnehmen, in der formalen C-Terminologie wird das Format auch UNSIGNED INT genannt. Das LONG-Format schließlich besteht aus einem 4-byte-Wort mit Vorzeichen. Der Wertebereich erstreckt sich von -2.147.483.648 bis +2.147.483.647.
Obgleich alle vier Integerformate recht unkompliziert zu sein scheinen, gibt es doch einige Besonderheiten. Zunächst einmal: Die allgemeine Definition der Sprache C erlaubt LONG und SHORT sowie vorzeichenlose und vorzeichenbehaftete Versionen der grundlegenden Datentypen. Bei dem hier zu besprechenden Compiler ist SHORT INT eine vorzeichenbehaftete 2-byte-Zahl (genau wie INT) und es gibt weder eine vorzeichenlose Version des 4-byte-LONG-Formates noch eine vorzeichenbehaftete Version des 1-byte-CHAR-Formates. Adressen erscheinen in Form des Datensegment-Offsets und werden im 16-bit-UNSIGNED-INT-Format gespeichert.

Die Beziehung zwischen Zeichen und Zahlen ist in C etwas verschwommen, das gilt insbesondere für CHAR und INT. Beim Lattice/Microsoft C-Compiler sind CHARs immer 1-byte-Zahlen mit einem Wert zwischen 0 und 255. CHAR kann aber auch als Zeichen ausgedrückt werden. So entspricht beispielsweise dem Buchstaben Q der ASCII-Wert 81. Wenn Sie in einem C-Programm aber die Zahl 81 schreiben, wird der Compiler das im INT-Format speichern, nicht als 1-byte-CHAR, es sei denn, Sie spezifizieren CHAR explizit. Da C, etwa im Gegensatz zu Pascal, keine starre Unterscheidung zwischen Zeichen und Zahlen kennt, lassen sich die Werte von Integerformaten (wie INT oder UNSIGNED INT) und Zeichenformaten (wie CHAR oder Stringelemente) ineinander umwandeln.

20.5.1.2 Stringdatenformate

Die Doppeldeutigkeit von Zeichen als Zeichen oder Zahlen schlägt sich in der Stringbehandlung nieder. Eine Zeichenkette (String) ist ein Feld aus CHARs, die durch das Zeichen Null beendet wird. Da die Null das Ende einer Zeichenkette anzeigt, ist klar, daß es in C keine Unterscheidung zwischen einem String fester Länge und einem String variabler Länge gibt. Um die Länge einer Zeichenkette festzustellen, muß also immer der gesamte String nach der Null abgesucht werden. Felder, die die Stringlänge bestimmen, gibt es in C im Gegensatz zu BASIC oder Pascal nicht.

Daraus folgt, daß auch keine Begrenzung in der Stringlänge existiert. Die Null, CHR$(0), kann in C nie Teil einer Zeichenkette sein, ein durchaus schwerwiegender Nachteil.
Das Nullzeichen wird vom C-Compiler automatisch an jedes Stringliteral angehangen. Wenn Sie in einem Programm beispielsweise "abc" schreiben, legt der Compiler eine Zeichenkette mit vier Bytes an, die ersten drei Zeichen sind "a", "b" und "c", das vierte Zeichen ist CHR$(0). Die Stringfunktionen erkennen das Nullzeichen als Stringende. Bedenken Sie in diesem Zusammenhang, daß das Nullzeichen nicht automatisch ein Teil des Datenformates ist, sondern ein Sonderzeichen des Compilers. Mit anderen Worten: Wenn Sie Routinen schreiben, die mit Strings umgehen und nicht auf die Standard-C-Funktionen zurückgreifen, ist es Aufgabe der Routinen, das Nullzeichen zu setzen und zu erkennen.

20.5.1.3 Fließkommadatenformate

Im Lattice/Microsoft C-Compiler existieren zwei Fließkommaformate: FLOAT und LONG FLOAT (oder DOUBLE). Das FLOAT-Format besitzt eine Länge von vier Bytes, das LONG FLOAT-Format verfügt über acht Bytes. Beide Formate entsprechen dem 8087/80287-Standardformat, das auch in vielen anderen Compilern (nicht nur C-Compilern) zu finden ist. Im IBM Pascal-Compiler sind die Formate als REAL4 und REAL8 implementiert und in dieser Form in Kapitel 20.4.1.4 ausführlich erklärt. Daher können wir uns an dieser Stelle auf einige wenige Hinweise beschränken.
Die Fließkommaformate sind am einfachsten zu erfassen, indem man sie als Bit-Kette mit 32 Bits (FLOAT) bzw. 64 Bits (LONG FLOAT) betrachtet. Die Speicherung erfolgt so, daß die höchstwertigen Bits im letzten Byte stehen.

Das erste, höchstwertige, Bit ist das Vorzeichen-Bit: 0 steht für ein positives, 1 für ein negatives Vorzeichen. Das nächste Feld aus acht (bei FLOAT) bzw. elf (LONG FLOAT) Bits spezifiziert den binären Exponenten in der verschobenen Form, die positive und negative Exponenten zuläßt und in diesem Kapitel schon erklärt wurde. Die Verschiebung beträgt bei FLOAT 127 und bei LONG FLOAT 1023. Um den echten Wert des Exponenten zu erhalten, sind von dem gespeicherten Wert 127 bzw. 1023 zu subtrahieren. Die restlichen Bits (23 oder 52) stellen die Nachkomma-Mantisse dar. Das (gedachte) Dezimalzeichen steht vor diesen Bits. In Kapitel 20.4.1.4 finden Sie eine Tabelle mit der Auflistung der einzelnen Felder (Vorzeichen, Exponent, Mantisse).

20.5.2 Assemblerschnittstellen in C

Die Regeln für den Aufbau einer Schnittstellenroutine, die in Kapitel 20.2.1 ausführlich erläutert werden, gelten auch für den Lattice/Microsoft C-Compiler. Dennoch wollen wir uns ein Beispiel für eine Schnittstelle in C ansehen. Das Programm stammt aus Kapitel 16.3 und ist von den hierfür unwesentlichen Elementen befreit worden. Es geht darum, daß aus einem gegebenen Datum der zugehörige Wochentag mit einer DOS-Routine berechnet werden soll.

```
 1                PGROUP GROUP PROG
 2      PROG      SEGMENT BYTE PUBLIC 'PROC'
 3                PUBLIC  WOCHENTAG
 4                ASSUME  CS:PROG
 5      WOCHENTAG PROC    NEAR
 6                PUSH    BP
 7                MOV     BP,SP
 8                ...
 9                POP     BP
10                RET
11      WOCHENTAG ENDP
12      PROG      ENDS
13                END
```

In Zeile 1 wird die Assemblerschnittstelle mit den LINK-Konventionen des Compilers koordiniert. Der Name PROG und die Klassifikation 'CODE' in Hochkommata in Zeile 2 sind ebenfalls erforderlich, um diesen Konventionen zu genügen.

Der Name jeder Routine (hier WOCHENTAG) muß als PUBLIC deklariert werden (Zeile 3), damit der Binder sie an Programme anbinden kann, die von Routinen aufrufen werden können. Die ASSUME-Anweisung ist Zeile 4 ist zum Assemblieren der NEAR-Prozedur nötig. Der C-Compiler führt externe Routinen als NEAR-Aufruf aus. Daher wird in Zeile 5 die Prozedur (PROC) als NEAR deklariert. Beachten Sie, daß das eine Abweichung von den allgemeinen Regeln für Schnittstellen ist.

Die Zeilen 6, 7, 9 und 10 dienen der Stapelmanipulation, die für jede Routine erforderlich ist.

Sie werden vielleicht bemerkt haben, daß hinter der RET-Anweisung in Zeile 10 kein Wert steht. Das ist die zweite Abweichung von den Standardregeln für Schnittstellen. Der Grund: Beim Lattice/Microsoft C-Compiler muß die Stapelleerung vom aufrufenden Programm aus geschehen, nicht von der aufgerufenen Routine. Daher ist hinter RET keine Angabe darüber nötig, wieviele Parameter-Bytes vom Stapel zu entfernen sind, wie das in den meisten anderen Programmiersprachen erforderlich ist.

Beachten Sie, daß in Standard-C und auch im Lattice/Microsoft-Compiler einer Unterroutine eine variable Anzahl von Parametern übergeben werden kann. Von dieser Möglichkeit wird z.B. bei PRINTF Gebrauch gemacht. Durch die Analyse von SP und BP kann von der Routine aus erkannt werden, wieviele Parameter im Stapel abgelegt wurden.
Die Zeilen 11 bis 13 sind Standard für das Ende einer Schnittstellenroutine.

20.5.3 Parameterübergabe in C

Parameter werden in C direkt oder indirekt in den Stapel gegeben. Der Lattice/Microsoft C-Compiler transferiert die zuletzt geschriebenen Parameter als erste in den Stapel, kehrt also die Schreibfolge praktisch um. Der erste Parameter wird als letztes in den Stapel genommen, er besitzt den niedrigsten Offset des Registers BP. Das ist wiederum eine Abweichung von der üblichen Handhabung in anderen Sprachen.
Wie bereits an anderer Stelle erklärt, können Parameter entweder direkt als Wert oder indirekt als Adresse im Stapel abgelegt werden. In C ist die Unterscheidung zwischen Wert und Adresse in einigen Fällen etwas schwierig, da sie eher einer akademischen Definition als dem gesunden Menschenverstand entspricht. So ist in C beispielsweise der Wert der Stringvariablen S die Adresse der Zeichenkette, nicht die Zeichenkette selbst. Anders ausgedrückt: Die Variable S hat als Wert eine Adresse. Ist S eine Stringvariable und I eine Integervariable und wird eine Routine mit

```
ROUTINE (S,I)
```

aufgerufen, werden die Werte von S und I in den Stapel gegeben; der Wert von I ist eine ganze Zahl, der Wert von S die Adresse eines Stringparameters.
Soll die Adresse eines Parameters übergeben werden, muß das Zeichen *Kaufmanns-Und* (&) vor den Variablennamen gesetzt werden:

```
ROUTINE (S,&I)
```

Bei diesem Aufruf werden zwei Adressen auf den Stapel geschoben: erstens die Adresse der Variablen I und zweitens die Adresse des Strings S. Beachten Sie, daß vor S kein *Kaufmanns-Und* steht, da bei einem String der Wert bereits die Adresse des Strings darstellt (siehe oben). Man gerät leicht in Versuchung `ROUTINE (&S,&I)` zu schreiben. Dadurch wird die Adresse der Adresse des Strings S in den Stapel geschoben wird. Wenn das beabsichtigt ist, ist der Aufruf richtig. In den meisten Fällen dürfte es sich allerdings um einen Fehler handeln. Das Zeichen & in C entspricht dem Wort VAR in Pascal, in beiden Fällen wird die Wahl einer Adresse erzwungen. Nach den Erfahrungen des Autors ist die Parameterübergabe

die Hauptfehlerquelle, wenn von einem C Programm aus eine Assemblerroutine aufgerufen wird.
Soll ein Parameterwert durch die Unterroutine geändert werden, legt das Hauptprogramm üblicherweise die Adresse des Parameters im Stapel ab, so daß die Routine über die Adresse auf den Wert zugreifen kann. Dabei ist zu beachten, daß in C Routinen mit NEAR aufgerufen werden, der Stapel-Offset also ein anderer als bei FAR-Aufrufen ist. Der erste Parameter liegt bei [BP+4], nicht bei [BP+6]. Dieser Parameter ist der beim Aufruf zuletzt geschriebene, wie bereits erwähnt.
Um Ihnen nun noch ein konkretes Beispiel des Parameterzugriffs in C zu geben, wollen wir annehmen, eine Routine sei mit zwei Integerzahlen als Parameter aufzurufen:

```
ROUTINE (&I,J)
```

Die Aufgabe der Routine sei es, den Wert des zweiten Parameters in den ersten zu übertragen, was der Zuordnung I = J gleichkommt. Die Routine sieht wie folgt aus (Ausschnitt):

```
MOV   AX,[BP+6]          ;Wert von J holen
MOV   BX,[BP+4]          ;Adresse von I holen
MOV   [BX],AX            ;Wert von J in I übertragen
```

In C wird üblicherweise von allen Routinen ein Integerwert in das AX-Register zurückgegeben, selbst wenn er vom Hauptprogramm nicht benötigt wird. Auch das ist wieder ein Punkt, in dem C sich von den meisten anderen Programmiersprachen unterscheidet, die eine klare Trennung zwischen Unterroutinen, die Werte zurückgeben und solchen, die das nicht tun, machen. Soll von einer Routine in C kein Wert an das aufrufende Programm übergeben werden, empfiehlt es sich, in AX den Wert 0 zu schreiben. Man kann darauf verzichten, wenn AX im Hauptprogramm nicht abgefragt wird.

20.6 Schlußbemerkung

Die Erläuterungen in diesem Kapitel beziehen sich nur auf einige wenige Programmiersprachen und nur auf Teilaspekte dieser Sprachen. Ein ausführlicher Vergleich der Sprachen würde den Rahmen dieses Buches sprengen. Die in diesem Kapitel vermittelten Kenntnisse dürften Ihnen dennoch nicht nur bei Assembler, BASIC, Pascal und C hilfreich sind, sondern auch bei der Beurteilung und beim Einsatz anderer Programmiersprachen, wenn Sie die erläuterten Sprachkriterien auf diese Sprachen anwenden.

Anhang A
Installierbare Schnittstellentreiber

A.1 Allgemeiner Überblick 362
A.2 Der ANSI-Treiber 363
A.2.1 ANSI-Bildschirmsteuerung 364
A.2.2 ANSI-Tastaturkontrolle 365
A.2.3 Vor- und Nachteile des ANSI-Treibers 365

Zwei Neuerungen, die mit den DOS-Versionen 2.00 und 2.10 eingeführt wurden, erfordern eine genauere Betrachtung: Die installierbaren Geräte- oder Schnittstellentreiber allgemein und der ANSI-Treiber (auch ANSI. SYS genannt) insbesondere. ANSI.SYS könnte man als Standard-Schnittstellentreiber bezeichnen.

A.1 Allgemeiner Überblick

DOS kann mit den geläufigen Peripheriegeräten wie Druckern, DFÜ-Einrichtungen (Modems), Diskettenlaufwerken und natürlich Tastatur und Bildschirm zusammenarbeiten. Zusätzlich kann eine Vielzahl anderer Geräte angeschlossen werden, vorausgesetzt, daß man in DOS Routinen einbindet, die die neuen Geräte unterstüzen. Die Routinen sind die Schnittstellentreiber, die eine Verbindung zwischen den Geräten einerseits und DOS oder DOS-Programmen andererseits herstellen.
Ab Version 2.00 lassen sich Schnittstellentreiber, die gewissen Regeln entsprechen, in die DOS-Operationen integrieren. Die Treiberprogramme werden von der Diskette geladen und während des Startprozesses von DOS aufgenommen. Die Datei CONFIG.SYS enthält die dafür erforderlichen Informationen. Name und Lage des Treibers werden durch die Befehlszeile *DEVICE=dateiname* in CONFIG.SYS spezifiziert. Zu jeder DEVICE-Programmzeile sucht DOS das entsprechende Programm, lädt es in den Speicher und durchläuft eine Reihe von Schritten, die zur Integration des Treibers notwendig sind.
Viele Geräte oder Schnittstellentreiber arbeiten nach den gleichen Prinzipien wie die DOS-Routinen für die Standardgeräte. Wird z.B. ein neues Disketten- oder Plattenlaufwerk angeschlossen, erfolgt die Unterstützung durch den zugehörigen Treiber im allgemeinen in gleicher Weise wie bei einem Standardlaufwerk, lediglich Details bezüglich der Befehle werden unterschiedlich sein. Schnittstellentreiber für eine Maus oder einen Joystick arbeiten zumeist ähnlich wie eine Tastaturunterstützung.
Andererseits können Schnittstellentreiber aber auch Funktionen erfüllen, die wenig oder gar nichts mit dem Anschluß neuer Geräte zu tun haben. Ein solcher Schnittstellentreiber ist ANSI.SYS, der die Operationseigenschaften vorhandener Geräte (Tastatur und Bildschirm) verändert.
Bevor wir uns dem ANSI-Treiber zuwenden, sollen Sie noch einige Hinweise zum Aufbau eines Treiberprogramms erhalten. Die Einzelheiten, die Sie wissen müssen, um selbst Treibersoftware zu schreiben, können hier nicht behandelt werden.
Eine Schnittstellentreiberdatei ist in ihrem Aufbau dem Format einer Programmdatei ähnlich, enthält aber Zusatzinformationen, die sie als Treiber ausweisen. Es gibt zwei Arten von Schnittstellentreibern: zeichen- und blockorientierte. Zeichentreiber sind für zeichen-orientierte Geräte wie Tastatur, Bildschirm, Drucker oder Kommunikationsport, die mit einem

seriellen Datenfluß arbeiten, geeignet. Blocktreiber unterstützen block-orientiert arbeitende Geräte, die wie ein Laufwerk Datenblöcke im wahlfreien Zugriff über eine Blockadresse lesen und schreiben. Zeichen-orientierte Geräte werden mit einem Namen bezeichnet (ähnlich den Namen LPT1: oder COM1:) und wie Dateien behandelt, während block-orientierte Geräte durch Laufwerksbuchstaben gekennzeichnet werden, die wie die Standardlaufwerksbuchstaben A, B, C usw. DOS zugeordnet sind.
Mehrere Einsprungadressen des Treiberprogramms müssen DOS bekannt sein, um den Treiber für verschiedene Zwecke aufrufen zu können, z.B. zur Initialisierung und zur Ausführung von Befehlen. Der Treiber muß die Standardkommandos, mit denen DOS auf jeden Treiber zugreift, korrekt interpretieren und ausführen können. Außerdem sollte der Treiber in der Lage sein, DOS mit Statusinformationen über das Gerät zu versorgen.
Einen Schnittstellentreiber zu erstellen kommt ungefähr der Aufgabe gleich, Ein-/Ausgaberoutinen auf DOS- oder ROM-BIOS-Ebene zu schreiben. Es gehört zu den schwierigsten Herausforderungen, die einem Programmierer begegnen können.

A.2 Der ANSI-Treiber

Ein Beispiel für einen installierbaren Schnittstellentreiber ist der ANSI-Treiber, der die Programmierung von Tastaturabfragen und Bildschirmausgaben erleichtert. In der IBM-Version von DOS wird der ANSI-Treiber nur aktiviert, wenn dies in CONFIG.SYS festgelegt ist. Beim Startprozeß überprüft DOS die Datei CONFIG.SYS und richtet sich entsprechend den dortigen Spezifikationen ein. Der Befehl, der in CONFIG.SYS enthalten sein muß, um den ANSI-Treiber zu laden, lautet:

```
DEVICE = ANSI.SYS
```

Für die Benutzung des ANSI-Treibers ist es gleichgültig, ob er erst mit CONFIG.SYS hinzugeladen wird (wie beim Original IBM-DOS) oder bereits ein fester Bestandteil des DOS ist (wie bei einigen IBM-kompatiblen Systemen).
Der ANSI-Treiber kontrolliert bzw. steuert alle Tastatureingaben und Bildschirmausgaben, die über die entsprechenden Standard-DOS-Routinen abgewickelt werden. Das bedeutet umgekehrt, das Daten, die an DOS vorbeigeleitet werden, nicht mit dem ANSI-Treiber bearbeitet werden können.
Wenn Daten über ANSI.SYS auf dem Bildschirm ausgegeben werden, unterscheidet der Treiber zwischen Codes, die dargestellt werden sollen und Codes, die Treiberbefehle sind. Codes, die der Treiber als Befehle erkennt, werden aus dem Datenfluß herausgenommen, so daß sie nicht auf dem Bildschirm erscheinen. Der Treiber arbeitet also als *Filter*, das für Bildschirmdaten durchlässig ist, während Treiberbefehle an den befehlsverarbeitenden Teil des Treibers weitergeleitet werden.

Die Befehle des Treibers sind durch einen speziellen 2-byte-Code gekennzeichnet: Das erste Byte beinhaltet den Wert hex 1B oder CHR$(27), das Zeichen *Escape*, das zweite Byte ist die öffnende Klammer "[", der Wert hex 5B oder CHR$(91). Auf die zwei Bytes folgen die Befehlsparameter und als letztes der Befehlscode. Die Befehlsparameter sind entweder Zahlen (in Form von numerischen ASCII-Zeichen, die als Zahlen interpretiert werden) oder Strings aus ASCII-Zeichen, die in Anführungszeichen eingeschlossen sein müssen. Falls mehrere Parameter erforderlich sind, müssen sie durch Semikolon getrennt sein. Der Befehlscode, der den Befehl abschließt, besteht aus einem einzigen Buchstaben. Die Groß-/Kleinschreibung darf dabei nicht vernachlässigt werden. "h" und "H" können zwei verschiedene Befehlscodes sein.
Nachfolgend finden Sie zwei Beispiele für Treiberbefehle. Der Stern steht für das Zeichen *Escape* (hex 1B):

```
*[1C
*[65;32;66;"Re-mapped B"p
```

Der ANSI-Treiber stellt eine Vielzahl verschiedener Befehle zur Verfügung, die sich in zwei Gruppen einteilen lassen: Bildschirm und Tastaturbefehle.

A.2.1 ANSI-Bildschirmsteuerung

Während auf ROM-BIOS-Ebene eine komplette Cursorsteuerung möglich ist, mit der der Cursor auf jede beliebige Bildschirmposition gesetzt werden kann, läßt DOS in dieser Hinsicht sehr zu wünschen übrig. Im Grunde arbeitet die DOS-Bildschirmausgabe wie bei einem Terminal oder einem Drucker. Ein Zeichen wird an das andere gesetzt, bis eine Zeile voll und ein Zeilenvorschub nötig ist; auf der neuen Zeile wird die Ausgabe in gleicher Weise fortgesetzt. Diese Methode wird dem enorm großen Leistungspotential des PC-Bildschirms nicht gerecht. Daher arbeiten die meisten Programmierer bei der Bildschirmsteuerung auf der BIOS-Ebene.
Die Bildschirmbefehle des ANSI-Treibers ändern diesen unbefriedigenden Zustand. Mit ihnen läßt sich das Leistungspotential des Bildschirms fast vollständig ausnutzen. Die Befehle ermöglichen z.B. Cursorbewegungen, Bildschirmlöschen, die Festlegung von Anzeigeattributen (z.B. Farbe, Unterstreichen, Blinken usw.) und das Wechseln zwischen Text- und Grafikmodi. Außerdem kann die Cursorposition gespeichert werden. Das ermöglicht es, den Cursor zu versetzen, Daten zu schreiben, und anschließend den Cursor wieder auf die Ursprungsposition zu bringen.

A.2.2 ANSI-Tastaturkontrolle

Neben den Bildschirmbefehlen stellt ANSI.SYS Tastaturumsetzungsbefehle zur Verfügung. Mit diesen Befehlen läßt sich erreichen, daß ANSI.SYS ständig die Tastatur abfragt und ankommende Zeichen je nach Spezifikation entweder in andere Zeichen oder Zeichenketten umwandelt. Das entspricht der Funktionsweise gängiger Tastaturmakroprogramme (siehe Kapitel 6).
Die Tastaturbefehle werden dem ANSI-Treiber ebenso wie die Bildschirmbefehle über eine Bildschirmausgabe übermittelt. Lassen Sie sich nicht dadurch irritieren, daß die Funktionen der Tastaturbefehle Eingabeoperationen sind.

A.2.3 Vor- und Nachteile des ANSI-Treibers

Die Befehlsliste des ANSI-Treibers kann von zwei unterschiedlichen Standpunkten aus betrachtet werden: aus der Perspektive des Anwenders und aus der des Programmierers. Aus Anwendersicht ist die Möglichkeit zur Bildung von Tastaturmakros das einzig Nützliche an ANSI.SYS, für den Programmierer ist ANSI.SYS ein Entwicklungswerkzeug. Nun ist das vorliegende Buch in erster Linie für Programmierer gedacht, aber ein guter Programmierer muß sich zwangsläufig auch die Anwendersicht zu eigen machen. Sie sollten die folgenden Sätze für bzw. über Anwender auf jeden Fall lesen.
Der ANSI-Treiber wird von Anwendern häufig als Tastaturerweiterung und als DOS-Befehlserweiterung angesehen. Mit den ANSI-Tastaturbefehlen lassen sich Makros bilden, so daß mit einem einzigen Tastendruck mehrere Tastenbetätigungen simuliert werden können. Üblicherweise schreibt man die Makrotastendrücken in eine Textdatei und läßt sie mit TYPE über den ANSI-Treiber auf dem Bildschirm ausgeben. Indem ANSI-Befehle in die Textdatei aufgenommen werden, läßt sich jeder beliebige Unsinn ausführen. Beispielsweise kann der Cursor an den oberen Bildschirmrand gesetzt, dort Uhrzeit und Datum in inverser Darstellung angezeigt und anschließend der Cursor wieder auf die Ausgangsposition zurückgesetzt werden. Oder man läßt den Bildschirm löschen und ein neues Menü erscheinen. Die Möglichkeiten sind unbegrenzt.
Aus der Sicht des Programmierers zeigt sich der ANSI-Treiber etwas anders. Die Verwendung des Treibers bringt mehrere Vorteile, davon zwei besonders wichtige. Programmierer, die nicht über das nötige Werkzeug und die Kenntnisse verfügen, die man braucht, um über Assemblerschnittstellen auf BIOS zuzugreifen, können die BIOS-ähnlichen Routinen des ANSI-Treibers von allen Programmiersprachen aus verwenden. Der zweite Vorteil ist, daß Programme, die den ANSI-Treiber anstelle von

BIOS-Routinen aufrufen, auf allen MS-DOS-Computern laufen, nicht nur auf den Original IBM PCs und vollständig kompatiblen.
Trotz dieser offensichtlichen Vorteile ist es grundsätzlich nicht ratsam, in Programmen mit den ANSI-Treiberbefehlen zu arbeiten. Der Grund: Die Programme sind nur auf Computern ablauffähig, auf denen der Treiber installiert ist. Es ist schon ohne ANSI.SYS schwierig genug, Anwendern zu erklären, wie der Computer und die Software zu benutzen sind. Der ANSI-Treiber verkompliziert die Materie zusätzlich.
Ein weiteres Argument gegen den Einsatz von ANSI.SYS in Programmen ist, daß der Treiber nicht unter allen Umständen verwendet werden kann. So verträgt er sich z.B. nicht mit "Topview", dem Fenstersystem von IBM. Programme, die die Anwesenheit des ANSI-Treibers erfordern, können nicht unter "Topview" genutzt werden. Das kann sich für andere Fenstersysteme genauso gut als wahr erweisen.
Das gewichtigste Argument gegen ANSI.SYS ist aber wohl die niedrige Geschwindigkeit, mit der der Treiber auf den Bildschirm ausgibt. Die Benutzung der BIOS-Routinen oder direktes Schreiben in den Bildschirmpuffer führen zu wesentlich höheren Ablaufgeschwindigketien. Mit dem NU-Programm der *Norton Utilities* (Version 3) läßt sich ein Geschwindigkeitsvergleich durchführen. Das NU-Programm enthält drei Bildschirmtreiber, die mit den drei genannten unterschiedlichen Methoden arbeiten (ANSI, BIOS, Bildschirmpuffer). Probieren Sie alle drei aus und Sie werden sehen, um wieviel langsamer der ANSI-Treiber seine Aufgabe erledigt. Für die Ausgabe größerer Datenmengen ist der ANSI-Treiber einfach zu langsam.

Anhang B
Hexadezimale Arithmetik

B.1 Bits und Hex 369
B.2 Segmentierte Adressen und Hex 370
B.3 Dezimal/Hexadezimal-Umwandlung 371
B.4 Verwendung von BASIC für hexadezimale Arithmetik 373
B.5 Hex-Addition 375
B.6 Hex-Multiplikation 376

Die hexadezimalen Zahlen tauchen aus einem sehr einfachen Grunde bei der Arbeit mit Computern immer wieder auf: Alle Computer arbeiten mit Binärzahlen und die Hexzahlen sind eine geläufige Art der Darstellung von Binärzahlen.
Hexadezimale Zahlen (oder einfach Hexzahlen) sind auf der Basis 16 aufgebaut, ähnlich wie unser dezimales Zahlensystem auf der Zahl 10 basiert. Der Unterschied besteht darin, daß Hexzahlen durch 16 Symbole ausgedrückt werden, während unsere dezimalen Zahlen mit zehn Symbolen (0, 1, 2,...,9) auskommen. Hexzahlen bestehen aus den zehn Ziffersymbolen (0 bis 9) und den Buchstaben A bis F. Die Bedeutung der Ziffersymbole bleibt erhalten ("fünf bleibt fünf"), die Buchstaben von A bis F entsprechen den Zahlen 11 bis 15.

Hex	Dez	Hex	Dez	Hex	Dez	Hex	Dez
0	Null	4	Vier	8	Acht	C	Zwölf
1	Eins	5	Fünf	9	Neun	D	Dreizehn
2	Zwei	6	Sechs	A	Zehn	E	Vierzehn
3	Drei	7	Sieben	B	Elf	F	Fünfzehn

Tabelle B-1 Die dezimalen Werte der 16 Hexziffern

Die Hexziffern A bis F werden normalerweise in Großbuchstaben geschrieben, manchmal sieht man aber auch kleine Buchstaben. Das ist ohne Bedeutung.
Das Hexadezimalsystem ist nach demselben Grundschema aufgebaut wie unser Dezimalsystem, lediglich die Basiszahl ist eine andere (16 statt 10). Die einzelnen Ziffern werden stellenweise gewichtet. Ein Beispiel zum Begriff *stellenweise wichten*. Schreiben wir die dezimale Zahl 123, meinen wir damit:

$$\begin{array}{rl} 1 \times 100 & (10 \times 10) \\ +\,2 \times 10 & \\ +\,3 \times 1 & \end{array}$$

Interpretieren wir 123 als Hexzahl, ergibt sich folgendes:

$$\begin{array}{rl} 1 \times 256 & (16 \times 16) \\ +\,2 \times 16 & \\ +\,3 \times 1 & \end{array}$$

Es gibt keine allgemein anerkannte Schreibweise, eine Hexzahl als Hexzahl zu kennzeichnen. In BASIC und bei anderen Gelegenheiten wird dafür die Vorsilbe &H verwendet, bei einigen Notationen muß das Zeichen "#" oder "16#" der Zahl vorangestellt werden. Meist wird jedoch einfach ein H hinter die Zahl geschrieben. Leider ist es auch weitverbreitet, die

Hexnotation überhaupt nicht zu kennzeichnen. Es wird dann vom Leser erwartet, daß er aus dem Zusammenhang erkennt, ob eine Zahl in dezimaler oder hexadezimaler Schreibweise zu verstehen ist. Zahlen in Listings sind diesbezüglich mit besonderer Vorsicht zu genießen. Prüfen Sie sorgfältig, ob eine Dezimalzahl oder eine Hexzahl gemeint ist. In diesem Buch können Sie die Hexzahlen an dem vorangestellten "hex" erkennen.
Wenn Sie mit Hexzahlen umgehen möchten, können Sie sich mit Hextabellen oder mit BASIC die Arbeit erleichtern. Bevor wir dazu kommen, sollen Sie noch erfahren, warum sich Binärzahlen so leicht in Hexschreibweise ausdrücken lassen und warum segmentierte Adressen praktisch immer in Hexnotation geschrieben werden.

B.1 Bits und Hex

Hexzahlen werden in erster Linie als "Abkürzungen" der Binärzahlen mit denen der Computer arbeitet, verwendet. Jede Ziffer umfaßt vier Bits einer binären Information. Im *binären Zahlensystem* (auch *Dualsystem*) mit der Basis 2 kann eine 4-stellige Zahl (vier Bits) 16 verschiedene Werte (Bit-Kombinationen) darstellen. Um eine solche Zahl mit einer Stelle vollständig zu erfassen (alle Zahlenwerte oder Kombinationen von 0 bis 15), braucht man ein Zahlensystem der Basis 16. Das ist der Grund, weshalb hexadezimale Arithmetik im Zusammenhang mit Computern so häufig angewandt wird.

Hex	Bit	Hex	Bit	Hex	Bit	Hex	Bit
0	0 0 0 0	4	0 1 0 0	8	1 0 0 0	C	1 1 0 0
1	0 0 0 1	5	0 1 0 1	9	1 0 0 1	D	1 1 0 1
2	0 0 1 0	6	0 1 1 0	A	1 0 1 0	E	1 1 1 0
3	0 0 1 1	7	0 1 1 1	B	1 0 1 1	F	1 1 1 1

Tabelle B-2 Die Bitmusterdarstellung für die 16 Hexziffern

Wenn Sie mit 2-byte-Worten arbeiten, erinnern Sie sich bitte, daß das niederwertige Byte vor dem höherwertigen im Speicher abgelegt wird. Lesen Sie hierzu auch in Kapitel 2.7 nach.

Bit	Wort	Byte	Wert Dez	Wert Hex
0	 1	 1	1	1
1	. . . . . 1 .	. . . . . 1 .	2	2
2	. . . . . 1 . .	. . . . . 1 . .	4	4
3	. . . . . 1 . . .	. . . . 1 . . .	8	8
4	. . . . . 1	. . . 1	16	10
5	. . . . . 1	. . 1	32	20
6	. . . . . 1	. 1	64	40
7	. . . . . 1	1	128	80
8	. . . . . 1		256	100
9	. . . . . 1		512	200
10	. . . . . 1		1.024	400
11	. . . . . 1		2.048	800
12	. . . . 1		4.096	1000
13	. . 1		8.192	2000
14	. 1		16.384	4000
15	1		32.768	8000

Tabelle B-3 Die hexadezimalen und dezimalen Äquivalente jedes Bits eines Bytes und eines 2-byte-Wortes

B.2 Segmentierte Adressen und Hex

Am häufigsten kommt das Hexadezimalsystem im Zusammenhang mit der Speicheradressierung zum Einsatz. Wie Sie vielleicht noch aus den Kapiteln 2 und 3 wissen, besteht eine vollständige Adresse aus 20 Bits oder 5 Hexziffern. Da der Mikroprozessor 8088 aber nur mit 16-bit-Zahlen arbeiten kann, werden Adressen in einen Segmentadreßteil und einen relativen Offset zerlegt. Die zwei Teile werden zusammen als 1234:ABCD geschrieben, wobei 1234 die Segmentadresse und ABCD den relativen Offset darstellt. Beide Teile sind in Hex anzugeben.
Der Segmentadreßteil wird behandelt, als ob er mit 16 multipliziert worden wäre, was einer Verschiedung der ganzen Zahl um eine Stelle nach links gleichkommt. Die letzte, freie Stelle wird mit einer 0 aufgefüllt. Wenn der verschobene Segmentadreßteil und der relative Offset addiert werden, erhält man die 20-bit-Adresse. Die segmentierte Adresse 1234: ABCD wird wie folgt in die vollständige Adresse umgewandelt:

```
  1 2 3 4 Ø      (beachten Sie die 0 an der rechten Seite)
+   A B C D
-----------
  1 C F 0 D
```

Mit dieser Methode können Sie jede segmentierte Adresse in eine 20-bit-Adresse umwandeln. Für die Addition nehmen Sie am besten die nachfolgenden Tabellen zu Hilfe.

Hinweis: Bedenken Sie, daß eine 20-bit-Adresse auf mehrere Arten segmentiert werden kann. Die Umwandlung einer segmentierten Adresse in eine vollständige Adresse liefert stets ein eindeutiges Ergebnis, die Umwandlung einer vollständigen in eine segmentierte Adresse nicht.

Am besten halten Sie sich bei der Konvertierung in eine segmentierte Adresse an folgendes Schema (Sie können aber grundsätzlich auch jedes andere verwenden): Betrachten Sie die erste Stelle der vollständigen Adresse mit drei folgenden Nullen als Segmentadreßteil, die übrigen vier Stellen der vollständigen Adresse stellen den relativen Offset dar. Nach dieser Methode würde die Adresse 1CF0D segmentiert als 1000:CF0D geschrieben. Im BIOS-Listing wird üblicherweise mit dieser Notation gearbeitet. Alle relativen Adressen, die dort erscheinen, haben einen (nicht ausgedruckten) Segmentadreßteil von F000.
Der Segmentadreßteil einer segmentierten Adresse ist in einem der Segmentregister enthalten und kann auf jede durch 16 teilbare Speicheradresse verweisen. Der relative Offset variiert je nach Anwendung. Code-Offsets in Programmsegmenten enthalten gewöhnlich den Wert hex 100, da 256 (hex 100) Bytes vor jedem Programm von dem Programmsegmentpräfix belegt werden. Daten-Offsets enthalten üblicherweise eine 0, während Stapel-Offsets zumeist hohe Zahlen beinhalten, da der Stapel innerhalb des Stapel-Segmentes sich adressenmäßig nach unten hin ausbreitet, also bei einer hohen Adresse beginnt.
Wenn Sie ein Beispiel für eine segmentierte Adresse sehen möchten, laden Sie einmal das DOS-Programm DEBUG in den Computer und starten Sie es. Sobald das Eingabezeichen "-" erscheint, geben Sie das Kommando D ein. Es wird ein Auszug aus dem Speicher aufgelistet. Die Adressen auf der linken Bildschirmseite werden im Standardformat segmentiert dargestellt.

B.3 Dezimal/Hexadezimal-Umwandlung

Mit den folgenden Tabellen wird es Ihnen leicht fallen, Zahlen zwischen dem Dezimal- und dem Hexadezimalsystem umzuwandeln. Der Wertebereich erstreckt sich von 0 bis 1.048.575 (hex FFFFF), das entspricht 1.024 Kbyte und deckt den Adreßbereich des PC vollständig ab.

fünfte Stelle von links				vierte Stelle			
Hex	Dez	Hex	Dez	Hex	Dez	Hex	Dez
. . . . 0	0	 8	8	. . . 0 .	0	. . . 8 .	128
. . . . 1	1	 9	9	. . . 1 .	16	. . . 9 .	144
. . . . 2	2	. . . . A	10	. . . 2 .	32	. . . A .	160
. . . . 3	3	. . . . B	11	. . . 3 .	48	. . . B .	176
. . . . 4	4	 C	12	. . . 4 .	64	. . . C .	192
. . . . 5	5	. . . . D	13	. . . 5 .	80	. . . D .	208
. . . . 6	6	. . . . E	14	. . . 6 .	96	. . . E .	224
. . . . 7	7	. . . . F	15	. . . 7 .	112	. . . F .	240

dritte Stelle				zweite Stelle			
Hex	Dez	Hex	Dez	Hex	Dez	Hex	Dez
. . 0 . .	0	. . 8 . .	2.048	. 0 . . .	0	. 8 . . .	32.768
. . 1 . .	256	. . 9 . .	2.304	. 1 . . .	4.096	. 9 . . .	36.864
. . 2 . .	512	. . A . .	2.560	. 2 . . .	8.192	. A . . .	40.960
. . 3 . .	768	. . B . .	2.816	. 3 . . .	12.288	. B . . .	45.056
. . 4 . .	1.024	. . C . .	3.072	. 4 . . .	16.384	. C . . .	49.152
. . 5 . .	1.280	. . D . .	3.328	. 5 . . .	20.480	. D . . .	53.248
. . 6 . .	1.536	. . E . .	3.584	. 6 . . .	24.576	. E . . .	57.344
. . 7 . .	1.792	. . F . .	3.840	. 7 . . .	28.672	. F . . .	61.440

erste Stelle von links			
Hex	Dez	Hex	Dez
0	0	8	524.288
1	65.536	9	589.824
2	131.072	A	655.360
3	196.608	B	720.896
4	262.144	C	786.432
5	327.680	D	851.968
6	383.216	E	917.504
7	458.752	F	983.040

Tabelle B-4 Stellenweise Umwandlung von Hexadezimal- in Dezimalzahlen und umgekehrt.

Zur Benutzung der Tabellen: Um beispielsweise die Hexzahl A1B2 in die dezimale Entsprechung umzuwandeln, zerlegen Sie die Zahl in ihre Stellen, lesen die zugehörigen Werte aus den Tabellen ab und addieren sie.

2	an der vierten Stelle wird zu	2
B	an der dritten Stelle wird zu	176
1	an der zweiten Stelle wird zu	256
A	an der ersten Stelle wird zu	40.960
	Die Summe beträgt	41.394

Der umgekehrte Vorgang, das Umwandeln von Dezimal- in Hexadezimalzahlen, ist ebenso einfach durchzuführen, aber schwieriger zu beschreiben. Wir wollen als Beispiel die Dezimalzahl 1.492 verwenden.
In der Tabelle für die erste Stelle (0 bis 983.040) suchen wir die höchste Dezimalzahl aus, die nicht über 1.492 liegt, das ist 0 (die nächsthöhere Zahl wäre bereits 65.536). Wir schreiben die zugehörige Hexziffer (ebenfalls 0) als höchste Stelle der gesuchten Hexzahl nieder und subtrahieren die Dezimalentsprechung von 1.492 (1.492 minus 0 ergibt 1.492). Mit dem Ergebnis der Subtraktion gehen Sie in die Tabelle für die zweite Stelle (0 bis 61.440) und wiederholen den Vorgang. Die untenstehende Auflistung zeigt, wie das vor sich geht. Die zu 1.492 zugehörige Hexzahl ist 005D4 oder (ohne führende Nullen) 5D4.

Stelle	Hexziffer	Wert	Dezimaler Verbleibende Dezimalzahl	
Beginn			1.492	
5	0	0	1.492	(= 1.492 0)
4	0	0	1.492	(= 1.492 0)
3	5	1.280	212	(= 1.492 – 1.280)
2	D	208	4	(= 212 – 208)
1	4	4	0	(= 4 4)
Ergebnis	005D4			

Tabelle B-5 Umwandlung der Dezimalzahl 1.492 in eine Hexzahl

B.4 Verwendung von BASIC für hexadezimale Arithmetik

Wollen Sie die Umwandlungen in BASIC durchführen, aktivieren Sie den BASIC-Interpreter und geben die entsprechenden Befehle im Direktmodus (ohne Zeilennummern) ein.
Um das dezimale Äquivalent einer Hexzahl zu erhalten, geben Sie z.B. für die Zahl 1234 folgendes ein:

```
PRINT &H1234
```

Beachten Sie, daß jede Hexzahl mit dem Präfix "&H" beginnen muß, um sie für BASIC als Hexzahl zu kennzeichnen. Um die Dezimalanzeige aufzubereiten, steht PRINT USING zur Verfügung:

```
PRINT USING "###,###,###"; &H1234
```

Wollen Sie eine Dezimalzahl in eine Hexzahl umwandeln, verwenden Sie die Funktion HEX$:

```
PRINT HEX$(1234)
```

Ging es bisher nur um die Umwandlung von Zahlen, wollen wir uns nun mit Hexarithmetik beschäftigen. Der BASIC-Interpreter ist dabei besonders nützlich, da man Dezimal und Hexzahlen beliebig mischen kann. Hier zwei Beispiele:

```
PRINT USING "###,###,###"; &H1000 - &H3A2 + 16 * 3
PRINT HEX$(1776 - 1492 + &H100)
```

Soll das Ergebnis weiterverwendet werden, etwa für weitere Berechnungen, wird man es einer Variablen zuweisen. Man erspart sich dadurch die erneute Eingabe des Wertes. Beachten Sie, daß Variablen, die Hexzahlen enthalten, als doppeltgenaue Variablen definiert werden sollten (# am Ende des Variablennamens), um ein Maximum an Genauigkeit zu erreichen. Hier ein Beispiel:

```
X# = 1776 - 1492 + &H100
PRINT USING "###,###,###"; X#, 2 * X#, 3 * X#
```

B.5 Hex-Addition

Die Addition von Hexzahlen erfolgt Stelle für Stelle, genau wie bei Dezimalzahlen. Die untenstehende Tabelle erleichtert das Summieren zweier Hexziffern. Suchen Sie in der Zeile die eine Hexziffer und in der Spalte die andere, im Schnittpunkt steht die Summe der beiden Hexziffern.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1		2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2			4	5	6	7	8	9	A	B	C	D	E	F	10	11
3				6	7	8	9	A	B	C	D	E	F	10	11	12
4					8	9	A	B	C	D	E	F	10	11	12	13
5						A	B	C	D	E	F	10	11	12	13	14
6							C	D	E	F	10	11	12	13	14	15
7								E	F	10	11	12	13	14	15	16
8									10	11	12	13	14	15	16	17
9										12	13	14	15	16	17	18
A											14	15	16	17	18	19
B												16	17	18	19	1A
C													18	19	1A	1B
D														1A	1B	1C
E															1C	1D
F																1E

Bild B-6 Addition zweier Hexziffern

B.6 Hex-Multiplikation

Die Multiplikation von Hexzahlen wird wie bei dezimalen Zahlen Stelle für Stelle durchgeführt. Mit der untenstehenden Tabelle können Sie sich die Arbeit erleichtern. Suchen Sie in der Zeile eine der beiden Hexziffern und in der Spalte die andere, das Ergebnis der Multiplikation finden Sie im Schnittpunkt.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1		1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2			4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3				9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4					10	14	18	1C	20	24	28	2C	30	34	38	3C
5						19	1E	23	28	2D	32	37	3C	41	46	4B
6							24	2A	30	36	3C	42	48	4E	54	5A
7								32	38	3F	46	4D	54	5B	62	69
8									40	48	50	58	60	68	70	78
9										51	5A	63	6C	75	7E	87
A											64	6E	78	82	8C	96
B												79	84	8F	9A	A5
C													90	9C	A8	B4
D														A9	B6	C3
E															C4	D2
F																E1

Bild B-7 Multiplikation zweier Hexziffern

Anhang C
Zeichen

C.1 Standard- und erweiterter Zeichensatz 378
C.1.1 Zeichenformat 381
C.1.2 Die ersten 32 ASCII-Zeichen 383
C.1.3 Umrandungszeichen 384
C.1.4 Grafik- und Blockzeichen 385
C.2 Formatierung von Textdateien 386
C.2.1 Standardtextdateiformate 386
C.2.2 Formate von Textverarbeitungsprogrammen 387

Der PC verfügt über 256 verschiedene Zeichen mit numerischen Codes von 0 bis 255. Die Codezuordnung der Zeichen kann mit der BASIC-Funktion CHR$ erfolgen, man spricht auch von den Zeichen CHR$(0) bis CHR$(255).
Der erste 128-Zeichen-Bereich von CHR$(0) bis CHR$(127) enthält die ursprünglichen Standard-ASCII-Zeichen, der zweite 128-Zeichen-Bereich von CHR$(128) bis CHR$(255) enthält Sonderzeichen als Erweiterung des Standard-ASCII-Zeichensatzes.
Die Standard-ASCII-Zeichen werden auf allen Computern weitgehend gleich behandelt, geringe Differenzen treten allenfalls bei den ersten 32 Zeichen auf. In Abschnitt C.1.2 werden wir näher darauf eingehen. Für die 128 Sonderzeichen besteht keine einheitliche Regelung, sie dienen je nach Computerhersteller oder Betriebssystem den unterschiedlichsten Zwecken.
Erfreulicherweise werden auf allen IBM PC-Modellen die gleichen Sonderzeichen verwendet. Bei kompatiblen Geräten hängt es vom Grad der Kompatibilität ab, ob der IBM-Zeichensatz zur Verfügung steht oder nicht. Sie sollten das vor allem bei Programmen beachten, die auch auf anderen PCs als den Original IBM Geräten verwendet werden sollen.

C.1 Standard- und erweiterter Zeichensatz

Sie sehen nachfolgend ein BASIC-Programm, das alle 256 Zeichen mit den numerischen Codes in Hex- und Dezimalnotation ausgibt. Sie können die Zeichen aber auch der entsprechenden Tabelle entnehmen.

```
1000 ' 'Zeichendarstellung
1010 '
1020 MONOCHROM = 1
1030 IF MONOCHROM THEN WW = 80 : HH = &HB000
     ELSE WW = 40 : hh = &HB800
1040 GOSUB 2000                    'DS-Register initialisieren
1050 FOR I = 0 TO 255              'Zeichencodes 0 bis 255
1060   GOSUB 3000                  'Informationen anzeigen
1070 NEXT I
1080 PRINT "Ende"
1090 GOSUB 6000
1095 SYSTEM
1999 '
2000 ' Initialisierung
2010 '
2020 DEF SEG = HH                  'DS-Register für POKE vorbereiten
2030 KEY OFF: CLS                  'Bildschirm löschen
```

```
2040 WIDTH WW : COLOR 14,1,1
2050 FOR I = 1 TO 25 : PRINT : NEXT I
2060 PRINT "Auflistung des Zeichensatzes"
2070 GOSUB 5000                                    'Unterueberschrift
2080 RETURN
2099 '
3000 ' 'Zeicheninformationen ausgeben'
3010 '
3020 PRINT USING " ###      ";I;
3030 IF I< 16 THEN PRINT "0";
3040 PRINT HEX$(I);"          ";
3050 POKE WW * 2 * 23 + 34, I                      'Zeichen einfügen
3060 GOSUB 4000                                    'Bemerkung drucken
3070 IF (I MOD 16) < 15 THEN RETURN                'Pause alle 16 Zeichen
3080 GOSUB 6000
3090 IF I < 255 THEN GOSUB 5000
3100 RETURN
3997 '
3998 ' 'Erläuterungen zu einzelnen Zeichen
3999 '
4000 IF I =   0 THEN PRINT "Leerzeichen";
4007 IF I =   7 THEN PRINT "Beep (Glocke)";
4008 IF I =   8 THEN PRINT "Rückwärtsschritt";
4009 IF I =   9 THEN PRINT "Tabulator";
4010 IF I =  10 THEN PRINT "Zeilenvorschuo";
4012 IF I =  12 THEN PRINT "Seitenvorschub";
4013 IF I =  13 THEN PRINT "Wagenrücklauf";
4026 IF I =  26 THEN PRINT "Textdateiende";
4032 IF I =  32 THEN PRINT "Leerzeichen";
4055 IF I = 255 THEN PRINT "Leerzeichen";
4997 PRINT                                         'Zeilenende
4998 RETURN
4999 '
5000 ' 'Unterueberschrift
5010 '
5020 COLOR 15
5030 PRINT
5040 PRINT
5050 PRINT "Dezimal – Hex – Zeichen – Erläuterung"
5060 PRINT
5070 COLOR 14
5080 RETURN
5999 '
6000 ' 'Pause
6010 '
```

```
6020 IF INKEY$ <> "" THEN GOTO 6020
6030 PRINT
6040 COLOR 2
6050 PRINT "Eine Taste drücken"
6060 COLOR 14
6070 IF INKEY$ = "" THEN GOTO 6070
6080 PRINT
6090 RETURN
```

Code (dez)	Code (hex)	Zeichen	Code (dez)	Code (hex)	Zeichen	Code (dez)	Code (hex)	Zeichen	Code (dez)	Code (hex)	Zeichen	Code (dez)	Code (hex)	Zeichen	Code (dez)	Code (hex)	Zeichen	Code (dez)	Code (hex)	Zeichen	Code (dez)	Code (hex)	Zeichen
0	00		32	20		64	40	@	96	60	`	128	80	Ç	160	A0	á	192	C0	└	224	E0	α
1	01	☺	33	21	!	65	41	A	97	61	a	129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
2	02	☻	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
3	03	♥	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	ú	195	C3	├	227	E3	π
4	04	♦	36	24	$	68	44	D	100	64	d	132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
5	05	♣	37	25	%	69	45	E	101	65	e	133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
6	06	♠	38	26	&	70	46	F	102	66	f	134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
7	07	•	39	27	'	71	47	G	103	67	g	135	87	ç	167	A7	º	199	C7	╟	232	E7	τ
8	08	◘	40	28	(	72	48	H	104	68	h	136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
9	09	○	41	29	)	73	49	I	105	69	i	137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
10	0A	◙	42	2A	*	74	4A	J	106	6A	j	138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
11	0B	♂	43	2B	+	75	4B	K	107	6B	k	139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
12	0C	♀	44	2C	,	76	4C	L	108	6C	l	140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
13	0D	♪	45	2D	-	77	4D	M	109	6D	m	141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
14	0E	♫	46	2E	.	78	4E	N	110	6E	n	142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
15	0F	☼	47	2F	/	79	4F	O	111	6F	o	143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
16	10	►	48	30	0	80	50	P	112	70	p	144	90	É	176	B0	░	208	D0	╨	240	F0	≡
17	11	◄	49	32	1	81	51	Q	113	71	q	145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
18	12	↕	50	32	2	82	52	R	114	72	r	146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
19	13	‼	51	33	3	83	53	S	115	73	s	147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
20	14	¶	52	34	4	84	54	T	116	74	t	148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
21	15	§	53	35	5	85	55	U	117	75	u	149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
22	16	▬	54	36	6	86	56	V	118	76	v	150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
23	17	↨	55	37	7	87	57	W	119	77	w	151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
24	18	↑	56	38	8	88	58	X	120	78	x	152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
25	19	↓	57	39	9	89	59	Y	121	79	y	153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
26	1A	→	58	3A	:	90	5A	Z	122	7A	z	154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
27	1B	←	59	3B	;	91	5B	[	123	7B	{	155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
28	1C	∟	60	3C	<	92	5C	\	124	7C	|	156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
29	1D	↔	61	3D	=	93	5D	]	125	7D	}	157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
30	1E	▲	62	3E	>	94	5E	^	126	7E	~	158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
31	1F	▼	63	3F	?	95	5F	_	127	7F	⌂	159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Tabelle C-1 Der Zeichensatz des IBM PC

In Zeile 1020 wird festgestellt, ob ein Monochrom- oder ein Farb-/Grafikadapter angeschlossen ist: 1 steht für Monochrom, 0 für Farbe/Grafik. Abhängig vom Wert der Zeile 1020 führt das Programm zwei Anpassungen durch: zum einen bezüglich des Bildschirmadreßbereiches, der mit POKE angesprochen wird, zum anderen bezüglich der Bildschirmbreite (40 oder 80 Spalten). Bei einem Farb-/Grafikadapter wird im allgemeinen der 40-Spalten-Modus bevorzugt, um eine klarere Darstellung zu erhalten. Die POKE-Anweisung in Zeile 3050 bewirkt, daß ein Zeichen auf dem Bildschirm dargestellt wird. Es muß POKE verwendet werden, weil einige Zeichen mit den normalen PRINT-Befehl nicht ausgegeben werden können. Eine Erklärung dafür finden Sie in Anschnitt C.1.2, in dem die ersten 32 Zeichen des Zeichensatzes behandelt werden.
Jedes der 256 verschiedenen Zeichen hat sein eigenes von den anderen Zeichen verschiedenes Erscheinungsbild, außer den Zeichen CHR$(0) und CHR$(255), die wie CHR$(32) als Leerzeichen erscheinen.

C.1.1 Zeichenformat

Zeichen, die auf dem Bildschirm erscheinen, werden durch Punkte in einer Matrixanordnung, die auch *Zeichenfeld* genannt wird, erzeugt. Es gibt zwei standardisierte Zeichenfelder, eines für den Monochromadapter (und kompatible Karten) und eines für den Farb-/Grafikadapter (und kompatible Karten). In beiden Fällen werden die Zeichen dargestellt, indem die entsprechenden Punkte in der Matrix erleuchtet werden.

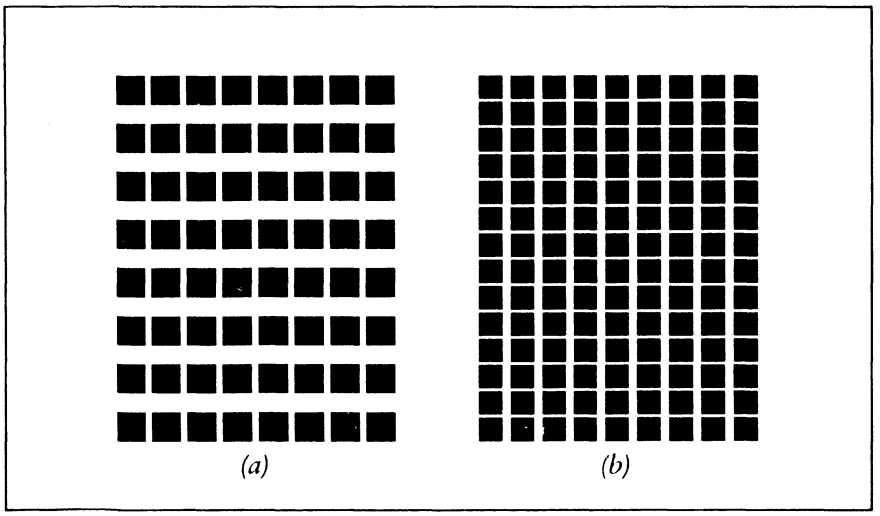


Bild C-1 Die Zeichenmatrix (a) des Farb-/Grafik- und (b) des Monochromadapters

Beim Farb-/Grafikadapter wird ein 8- mal 8-Punkte-Zeichenfeld verwendet, beim Monochromadapter ein 9 mal 14-Punkte-Zeichenfeld. Daher erlaubt der Monochromadapter eine feinere Zeichendarstellung als der Farb-/Grafikadapter, was sich z.B. bei Textverarbeitung sehr angenehm bemerkbar macht.
Bei Punktmatrixdruckern werden die Zeichen ebenfalls in einer Punktmatrix aufgebaut. Das bedeutet aber nicht, daß die Zeichen des Druckers exakt mit denen des Bildschirms übereinstimmen müssen. Es kann also zu einem veränderten Aussehen kommen. Besonders unerfreulich ist das bei Grafiken, die gemäß der Bildschirmvorgabe widergegeben werden sollen. Mangelnde Kompatibilität kann hier sehr schnell zu einem Problem werden.
Um Ihnen einmal den Zeichenaufbau zu zeigen, sehen Sie in nachstehender Abbildung ein "Y", ein "y" und ein Semikolon in einem 8- mal 8-Punkte-Feld dargestellt.

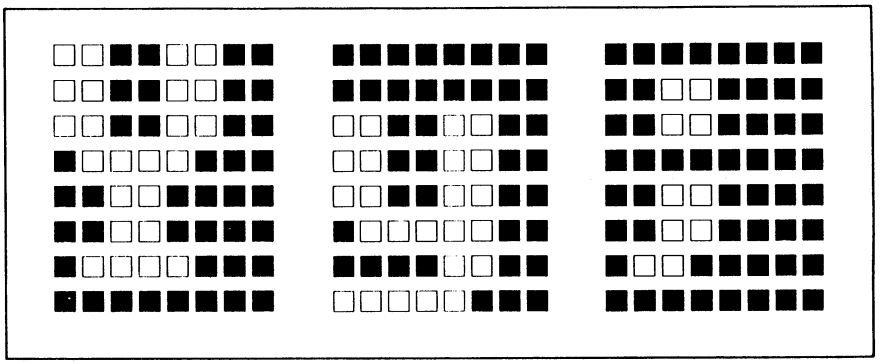


Bild C-2 Das Pixelbild von drei Zeichen in einem 8 × 8-Zeichenfeld

Es gibt mehrere Regeln, nach denen der Zeichensatz aufgebaut ist. Bei normalen Zeichen bleiben die zwei Spalten an der rechten Seite ungenutzt, sie dienen als Trennung zwischen den einzelnen Zeichen. Die zwei Spalten werden nur bei Zeichen verwendet, die das gesamte Zeichenfeld ausfüllen wie z.B. CHR$(219). Die zwei obersten Reihen werden nur bei Zeichen, die über die normale Zeichenhöhe hinausragen, belegt; das sind z.B. die Großbuchstaben oder Kleinbuchstaben wie "b", "k" oder "h". Die unterste Zeile des Zeichenfeldes ist für Zeichen, die unter die Schreiblinie gehen, vorbehalten. (z.B. "g" oder "y"). Im Rahmen eines optimalen Gesamterscheinungsbildes werden bei einigen Zeichen Kompromisse geschlossen, die von den Regeln abweichen. So ist beispielsweise das Semikolon eine Reihe angehoben (es belegt nicht die unterste Zeile des Zeichenfeldes), weil es sich in dieser Position besser in den Text einfügt.

Bits								Wert
7	6	5	4	3	2	1	0	(hex)
1	1	0	0	1	1	0	0	CC
1	1	0	0	1	1	0	0	CC
1	1	0	0	1	1	0	0	CC
0	1	1	1	1	0	0	0	78
0	0	1	1	0	0	0	0	30
0	0	1	1	0	0	0	0	30
0	1	1	1	1	0	0	0	78
0	0	0	0	0	0	0	0	00

Tabelle C-2 Die Kodierung der acht Zeichen-Bytes des Buchstabens „Y"

In den Grafikmodi 4 bis 6 und 10 können Sie anhand der genannten Regeln eigene Zeichen entwerfen. Die Zeichenaufbautabellen verwenden eine Kodierung von acht Bytes pro Zeichen - für jede Zeile des Zeichens ein Byte. Die acht Bits jedes Bytes bestimmen, welche Punkte gesetzt werden. Das *Y* hat z.B. folgende Kodierung: CC CC CC 78 30 30 78 00. Die einzelnen Bits in jedem Byte sehen Sie in folgender Tabelle. Wenn Sie genau hinschauen, erkennen Sie das *Y* in Form der Einsen wieder:

C.1.2 Die ersten 32 ASCII-Zeichen

Die ersten 32 ASCII-Zeichen, CHR$(0) bis CHR$(32), dienen zwei wichtigen Anwendungen, die manchmal miteinander in Konflikt stehen. Auf der einen Seite haben diese Zeichen standardisierte ASCII-Bedeutungen, etwa zur Druckersteuerung (z.B. CHR$(12) bedeutet Seitenvorschub) und zur Kommunikationskontrolle. Gleichzeitig existieren im IBM PC so interessante und nützliche Zeichen wie z.B. die Kartensymbole (Herz, Karo, Kreuz und Pik), CHR$(3) bis CHR$(6).
Generell reagieren alle Geräte, auch Drucker und Bildschirm, auf die ASCII-Bedeutung eines Zeichens und drucken das Zeichen nicht aus bzw. stellen es nicht dar. Ein Beispiel ist CHR$(7), die *Glocke*, die als Bild ein Punkt in der Mitte des Zeichenfeldes ist. Wenn Sie in BASIC

```
PRINT CHR$(7)
```

programmieren, erzeugt der Lautsprecher einen Glockenton. Wird das Zeichen aber mit dem POKE-Befehl direkt in den Bildschirmpuffer gebracht, erscheint die Zeichendarstellung auf dem Schirm:

```
DEF SEG = &HB800 : POKE 0, 7
```

Mit POKE lassen sich alle Zeichen auf dem Bildschirm darstellen. Gegenüber PRINT ergibt sich allerdings der Nachteil, daß Programme mit POKE schwerer an unterschiedliche Betriebsbedingungen anzupassen sind.

Es ist im allgemeinen besser, die Bildschirmausgabe mit den Standardmethoden wie z.B. PRINT in BASIC vorzunehmen.
Die meisten der ersten 32 Zeichen können auf dem Bildschirm dargestellt werden, je nach Programmiersprache können sie aber ein unterschiedliches Aussehen annehmen. In der nachfolgenden Tabelle finden Sie einige Beispiele. Die nicht aufgeführten Zeichen, CHR$(0) bis CHR$(6) und CHR$(14) bis CHR$(27), sehen in allen Programmiersprachen gleich aus.

Zeichen	BASIC	Resultat in den meisten anderen Programmiersprachen
CHR$(7)	Tonerzeugung	Tonerzeugung („Glocke")
CHR$(8)	Zeichen	Rückschritt ("Backspace")
CHR$(9)	Tabulator	Tabulator
CHR$(10)	Zeilenvorschub und Wagenrücklauf	Zeilenvorschub
CHR$(11)	Cursor nach links oben setzen	Zeichen
CHR$(12)	Bildschirm löschen	Zeichen
CHR$(13)	Wagenrücklauf	Wagenrücklauf
CHR$(29)	Cursor nach links	Zeichen
CHR$(30)	Cursor nach oben	Zeichen
CHR$(32)	Cursor nach unten	Zeichen

Tabelle C-3 Bedeutung bestimmter Zeichen in verschiedenen Programmiersprachen bei der Bildschirmausgabe („Zeichen" bedeutet, daß das Zeichenabbild auf dem Bildschirm erschient)

C.1.3 Umrandungszeichen

Die nützlichsten Sonderzeichen sind diejenigen, mit denen sich Umrandungen (mit einfachen oder doppelten Strichen) anfertigen lassen. Die Zeichencodes sind CHR$(179) bis CHR$(218). Die folgende Übersichtsdarstellung wird Ihnen beim Entwerfen von Rahmen und ähnlichem hilfreich sein.

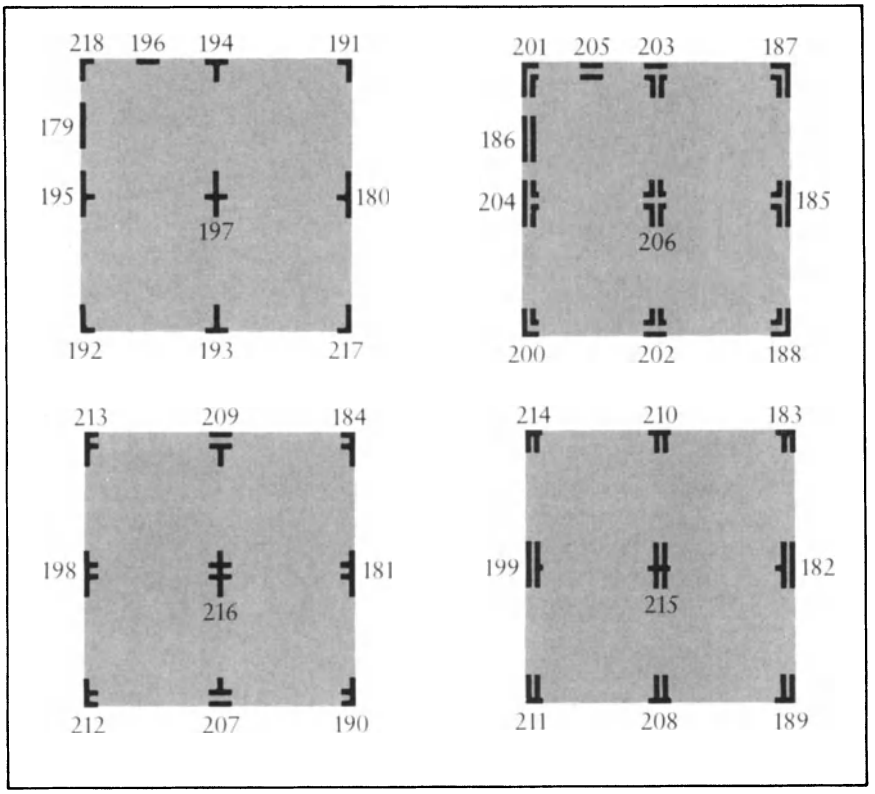


Bild C-3 Die Grafikzeichen für Umrandungen

C.1.4 Grafik- und Blockzeichen

Zusätzlich zu den Umrandungszeichen stehen Ihnen zwei weitere Zeichenserien zur Verfügung, die für Diagramme und Blockzeichnungen verwendbar sind. Die vier Zeichen der einen Serie erfüllen in unterschiedlichen Schattierungen das gesamte Zeichenfeld; bei den drei heller schattierten Zeichen sind nicht *alle* Punkte des Zeichenfeldes erleuchtet bzw. auf die Vordergrundfarbe gesetzt. Die andere Serie besteht aus Blockzeichen, bei denen nur die Hälfte des Feldes genutzt wird, dies aber mit der größtmöglichen Farbintensität (alle Punkte erleuchtet bzw. auf Vordergrundfarbe gesetzt). CHR$(219), das ein Zeichenfeld mit maximaler Intensität vollständig ausfüllt, paßt sich in beide Zeichenserien ein.

CHR$(176)	▒	CHR$(220)	▄
CHR$(177)	▓	CHR$(221)	▌
CHR$(178)	▓	CHR$(222)	▐
CHR$(219)	▉	CHR$(223)	▀

Bild C-4 Die Blockgrafikzeichen: gerastert (links) und geteilt (rechts)

C.2 Formatierung von Textdateien

Viele Programme arbeiten mit Dateien, die Text enthalten. In Folge dessen haben sich im Laufe der Zeit einige Konventionen bezüglich der Formate von Textdateien herausgebildet, an die sich die meisten Programmierer halten. Dadurch wird der Datenaustausch zwischen unterschiedlichen Programmen erleichtert. Die Formate werden durch Zeichen bestimmt, die im Text enthalten sind und Funktionen wie Wagenrücklauf (*Carriage Return*), Zeilenvorschub (*Line Feed*) oder Rückschritt (*Backspace*) ausführen.
Es ist sinnvoll, Programme so zu gestalten, daß sie verschiedene Textformate verarbeiten können. Andererseits ist es aber auch wichtig, die Formatierungen von Texten einfach zu halten und nicht zu viele Formatzeichen zu verwenden. Bei Textverarbeitungsprogrammen ist das nicht immer möglich, da hier sehr viele Textformatierungsmöglichkeiten zur Verfügung gestellt werden müssen (auch Änderungen schon vorhandener Formatierungen). In diesem Abschnitt wollen wir uns erst die Standardtextdateiformate und anschließend die Formate von Textsystemdateien ansehen.

C.2.1 Standardtextdateiformate

Normale Textdateien sind aus den Standard-ASCII-Zeichen aufgebaut und enthalten keine Sonderzeichen über CHR$(127). Im ASCII-Kodierungsschema sind den Zeichen CHR$(0) bis CHR$(32) besondere Bedeutungen zugeordnet. Einige werden zur Formatierung einer Textdatei verwendet, andere dienen der Übertragungssteuerung. Die Zeichen, die zur Formatierung herangezogen werden, werden nicht auf dem Bildschirm dargestellt.
Im allgemeinen sind es nur wenige Zeichen, die in normalen Textdateien zur Formatierung herangezogen werden. Ursprünglich waren diese Zeichen zur Druckersteuerung gedacht, heute können sie bei fast allen Ausgabegeräten verwendet werden. Die wichtigsten Formatzeichen finden Sie nachfolgend erläutert.

CHR$(26) zeigt das *Textende* einer Textdatei an. Das Zeichen kann schon erscheinen, bevor die Dateilänge, die im Verzeichniseintrag steht, erreicht ist. Textverarbeitungsprogramme lesen und schreiben Dateien am allgemeinen nicht Byte für Byte, sondern in größeren Einheiten - meist 128 Bytes auf einmal. Werden Daten auf diese Weise transferiert, kann DOS nur das Ende des 128-Byte-Blockes, nicht das tatsächliche Textende erkennen. Daher muß dieses durch CHR$(26) gekennzeichnet werden.
CHR$(13) und CHR$(10) unterteilen einen Text in Zeilen, indem das Ende jeder Zeile durch einen *Wagenrücklauf* (CHR$(13)) und einen *Zeilenvorschub* (CHR$(10)), im allgemeinen in dieser Reihenfolge, gekennzeichnet wird. Bei viele Textverarbeitungsprogrammen gibt es Schwierigkeiten mit Zeilen, die mehr als 255 Zeichen beinhalten, manchmal liegt die Grenze sogar bei 80 Zeichen pro Zeile.
Die Bedeutung des Zeichens *Wagenrücklauf* ist nicht ganz eindeutig: Manchmal schließt das Zeichen den Zeilenvorschub mit ein, manchmal nicht. Programm und Ausgabegerät (z.B. Drucker) müssen diesbezüglich abgestimmt sein, damit es keine Schwierigkeiten gibt.
CHR$(4) ist das *Tabulatorzeichen*, es wird manchmal verwendet, um ein oder mehrere Leerzeichen bis zur nächsten Tabulatorposition zu erzeugen. Es gibt keine allgemein gültige Norm für das Setzen von Tabulatorpositionen, die Verwendung des Zeichens kann daher zu undefinierten Abständen führen. Die wohl gebräuchlichste Tabulatoreinstellung ist ein Abstand von acht Leerzeichen.
CHR$(12), der *Seitenvorschub* oder *Seitenumbruch*, ist ein Formatzeichen, das ein Drucker als Papiervorschub zum Anfang einer neuen Seite interpretiert.
Es gibt auch noch andere Formatzeichen wie z.B. den vertikalen Tabulator CHR$(11), die aber auf PCs kaum Verwendung finden.
Viele Schwierigkeiten lassen sich vermeiden, wenn man Programme mit einer einfachen Dateiformatierung schreibt. Das einfachste Format erlaubt Zeilen mit einer maximalen Länge von 255 Zeichen, und kennt nur die Sonderzeichen Wagenrücklauf (CHR$(13), Zeilenvorschub (CHR$(10) und Dateiende (*End of File* EOF; CHR$(26)). Die meisten Programmiersprachen wie beispielsweise BASIC oder Pascal setzen die Formatzeichen automatisch, wenn ein Text eingegeben wird. Normalerweise verarbeiten sie die Zeichen auch selbsttätig und Sie müssen sich mit der Textformatierung nur auseinandersetzen, wenn Sie die Datenkontrolle der Programmiersprache umgangen haben.

C.2.2 Formate von Textverarbeitungsprogrammen

In Textverarbeitungsprogrammen kommen meist Sonderzeichen für Formatierungszwecke zum Einsatz. Dateien, die von solchen Programmen angelegt werden, sind selten einfach gehalten und haben meist viele exoti-

sche Erweiterungen gegenüber dem einfachen ASCII-Format. Generell hat zwar jedes Textprogramm seine eigenen Formatierungsregeln, es gibt aber dennoch einige Standards, die häufig anzutreffen sind.
Viele spezielle Formatcodes werden durch die erweiterten ASCII-Codes verwirklicht, die um den Wert 128 höher sind als der normale ASCII-Code. Das entspricht einem gesetzten höchstwertigen Bit (Bit 7). So wird z.B. der Soft-Wagenrücklauf CHR$(141) kodiert, indem zum normalen Wagenrücklauf der Wert 128 addiert wird. Es ergibt sich also 13 + 128 = 141, da CHR$(13) der Code für Wagenrücklauf ist. Ein Soft-Wagenrücklauf wird oftmals als vorläufiges Zeilenende verwendet, das verändert werden kann, wenn der Absatz neu formatiert wird. Andererseits kann der normale Wagenrücklauf das Ende eines Absatzes anzeigen, der durch erneutes Formatieren nicht verändert wird. Diese Art der Kodierung kann manche Programme dazu bringen, ganze Absätze als eine einzige Zeile zu behandeln.
Soft-Bindestriche, CHR$(173), deren Code um den Wert 128 höher ist als der normaler Bindestriche, CHR$(45), werden manchmal verwendet, um eventuelle Worttrennungen am Ende einer Zeile anzuzeigen. Gewöhnliche Bindestriche, CHR$(45), werden als reguläre Zeichen behandelt und können durch Textprogramme nicht automatisch verlegt werden, wie das bei Soft-Bindestrichen der Fall ist.
Sogar normale alphabetische Zeichen können verändert werden, indem 128 zu ihrem ursprünglichen Code addiert wird. In manche Programmen wird dadurch der letzte Buchstabe eines Wortes gekennzeichnet. Beispiele: Das kleine *a* hat den Code CHR$(97), wenn es aber am Ende eines Wortes erscheint, wie z.B. in *Kanada*, wird es vielleicht (je nach System) als CHR$(225) gespeichert, da 255 die Summe aus 97 und 128 ist. Der Vorteil dieser Methode ist, daß man Wortzwischenräume aus einem Leerzeichen nicht zu speichern braucht, sondern bei der Ausgabe nach jedem Endbuchstaben einschieben kann.
Wenn Sie ein Programm schreiben, das eine Vielzahl von Textdateiformaten lesen muß, sollten Sie die Sonderzeicheninterpretation variabel gestalten, so daß sie leicht (z.B. per Menü) an unterschiedliche Gegebenheiten angepaßt werden kann.

Sachwortverzeichnis

Adresse 14
Adreßinterrupts 47
    Endadresse 235
    Fehlerbehandlungsroutine 236
    Unterbrechungsadresse 235
Alt-Taste (siehe Tastatur)
ANSI-Treiber 282, 362, 365
ANSI.SYS 362
Arbeitsregister (siehe AX-, BX-, CX- und DX-Register)
ASCII-Zeichen 378
ASCIIZ-Strings 281
Assembler (siehe auch Schnittstellenroutinen) 20, 323, 331
    Befehlssatz des 8088 21
    CALL FAR bzw. NEAR 152
    Gerüst einer Routine 332
    Regeln 33
    Schnittstelle 333
AX-Register 27

BASIC 30, 330
    Assemblerschnittstelle 341, 344
    Datenformate 336
    Hexarithmetik 373
    Interpretierend 331, 339, 341
    Kompilierend 331, 340, 344
    SCREEN-Anweisung 71
BASICA (siehe BASIC)
Basiszeigerregister (BP) (siehe Offsetregister)
Baustein 2
    Speicherbaustein 13
    Support-Chip 8
Beenden, aber im Speicher verbleiben 285
Beenden eines Programmes 251
Befehlszeiger (IP) (siehe Offsetregister)
Bibliothek (siehe Objektcode)

Bildschirm
- Adapter
  - HR-Farbgrafikadapter - EGA 67, 70, 78
  - Original-Farb-/Grafikadapter 67, 70, 78, 87
  - Monochromadapter 67, 70, 78, 86
- ANSI.SYS 362
- AT 156
- Auflösung 70
- Ausgabe 252
- Bildschirmbreite 55
- Bildschirmspeicher 79, 80
- Bildschirmseiten (Anzeigeseiten) 80, 160
- Bildschirmsteuerung 84, 364
- BIOS-Routinen 84, 156
- Direkte Hardwarekontrolle 86
- Farbe 72
  - Farbunterdrückung 74
  - Intensität 72
  - Pixel verändern 188
  - RGB 72
- Farbpaletten 72, 165
  - 4-Farben-Palette 78
  - 16-Farben-Palette 72, 73, 78
  - 64-Farben-Palette 73
  - 256-Farben-Palette 73
- Flimmern 69
- Kompatibilität 89
- Memory-Mapped 68
- Modi 69, 71, 167
  - Farbunterdrückende 70, 74
  - Grafikmodi 66
    - BASIC 75
    - Bildschirmseiten 81
    - Farben 77
    - Pixel 83
    - Speichererfordernisse 80
    - Zeichenerzeugung 82
  - Textmodi 66
    - Attribute 76
    - Blinken 75
    - Farbkontrolle 75, 76
    - Bildschirmseiten 80
    - Monochrommodus 77
    - Speicherbezogene Zeichendarstellung 68, 82
    - Zeichenerzeugung 82
    - Zeichenattribute

Pixel 68
Rasterscan 68
Register 157
Seiten 80
Speicher 79
Vertikaler Rücklauf 69
Binden (siehe Linken)
BIOS (Basic Input/Output System)
Allgemeine Erläuterungen 16, 146, 148
Assemblerschnittstelle 149
Ausstattungsliste 202
Bildschirmdruckroutine 202
Bildschirmroutinen 157
Interrupts 147, 148
Laufwerksroutinen 172
Register 149
ROM-BASIC 204
Speicherkapazität feststellen 203
Tageszeitroutine 205
AT 206
Urladerstartroutine 204
Zählerstand lesen/setzen 206
Zusammenfassung 208
Bit-Mapped (Anzeige) 82
Boot-Eintrag (siehe Urlader)
BP - Basiszeigerregister (siehe Offsetregister)
Break 236
Bus 11
Adreßbus 12
Datenbus 12
Erweiterungsanschlüsse 11
Kontrollbus 11
8-bit-/16-bit-Bus 12
BX-Register 28

C - Programmiersprache
Allgemeine Erläuterungen 325, 331, 355
Assemblerschnittstellen 358
Datenformate 255
Parameterübergabe 359
Cartridge 62
Chip (siehe Baustein)
Cluster 115
Fragmentierung 116
Numerierung 117
Codesegment (siehe CS-Register)

.COM-Dateien
    Umwandlung zu .EXE-Dateien 324
COMMAND.COM 45
Compiler 323
CONFIG.SYS 362
CRC (siehe Prüfsummenfehler)
CRT Controller (siehe 6845) 66
CS-Register 29
Ctrl-Alt-Del (siehe Tastatur)
Ctrl-Break (siehe Tastatur)
CX-Register 28
Cursor 92, 158, 159

Dateiattribute (siehe Unterverzeichnis)
Dateien 258, 291, 305
Dateikontrollblock (siehe FCB)
Dateiverzeichnis (siehe Verzeichnis)
Dateiein-/ausgabe (siehe Ein-/Ausgabe)
Dateigruppenzeichen 268
Dateinamen (siehe Verzeichnis)
Dateinamenserweiterung (siehe Verzeichnis)
Dateinummern 281, 296
Dateisharing in DOS-3 291, 296
Dateisuche 301
Dateiverzeichnis (siehe Verzeichnis)
Dateizeiger 293
Datenbereich 115
Datenformat des 8088 24
Datensatz 260, 264, 266
Datensegmentregister (siehe DS-Register)
Datum (siehe Zeit- und Datumsfunktion)
DI (siehe Offsetregister)
Diskettenlaufwerke (siehe Laufwerk)
Diskettenein-/-ausgabe (siehe Ein-/Ausgabe)
Diskettentransferbereich (DTA) 263
Disketten
    BIOS-Routinen 172
    DMA - direkter Speicherzugriff (siehe 8237A)
        AT 9
        DMA-Controller 9
    DOS-Routinen (siehe Ein-/Ausgabe)
    Fehlerstatus 54, 173
    Formate 98
        Quad-Densitiy 99
        Standard-DOS 98

Kapazität 286
Markieren schlechter Spuren 117
Logische Struktur 108
Parameter 109
Physikalische Struktur 96
Dichte 96
Indexlöcher 177
Softsektoriert 96
Spuren 96
Sektoren 104, 105
Speicherbelegung
Dateiverzeichnis 106
Dateizuordnungstabelle (FAT) 106, 117, 120
Datenspeicherbereich 107
Urladereintrag 106, 108
Verifikation 270
DOS
Allgemeine Erläuterungen 222
Bildschirm 223
Dateien
COMMAND.COM 44
IBMBIO.COM 44
IBMDOS.COM 44
Diskettenformate (siehe Disketten) 225
DOS-2-Erweiterungen 279, 283
DOS-3-Erweiterungen 303
Endadresse 235
Fehlerbehandlungsadresse 236
Fehlercodes 231, 280, 303
Floppy- und Plattenstation 223
Interrupt
Adreßinterrupts 234
Druckerspoolerkontrolle 233
Hauptinterrupts 229
Zusammenfassung 228
Kompatibilität 225
Unterbrechungsadresse 235
Versionen 224, 283
Zusammenfassung 310
Druckroutinen 200
BIOS-Routinen 200
DOS-Routinen 253
DS-Register 29
DTA (Diskettentransferbereich) 263, 283
DX-Register 28

E/A (siehe Ein-/Ausgabe)
Ein-/Ausgabe 294
BIOS
Bildschirmroutinen 162
DOS
Ausgabe, seriell 253
Eingabe, seriell 252
Ein-/Ausgabeports (siehe Ports)
End-of-File-Marke 387
EGA - HR-Farb-/Grafikadapter (siehe Bildschirm)
ES-Register 29
.EXE-Format
Konvertierung zum .COM-Format 324
LINK 324
EXEC-geladene Programme 298
Erweiterte DOS-Funktionen 278
Extrasegmentregister (siehe ES-Register)

Farb-/Grafikadapter (siehe Bildschirm)
FAT (Dateizuordnungstabelle) 116, 120
Anzahl der Kopien auf der Diskette 116
Aufteilung der Diskette 116
Belegungskette 118
Beschädigungen 117
Clusterzuteilung 118
Dekodierung des FAT-Wertes 118
DOS-Routinen
FAT-Informationen 119, 263
16-bit-Format 119
12-bit-Format 119
FCB (Dateikontrollblock) (siehe auch E/A) 270
Allgemeine Erläuterungen 270
Dateigröße in Byte 270
Datensatzlänge 271
Erweiterung 271
Feldbeschreibungen 271
Öffnen von Dateien 258
FDC (Floppydisk-Controller) (siehe PD765)
Fehlerbehandlung (kritisch) 236
Fehlercodes 280, 299, 304
Fenster rollen 161, 162
Fenstersysteme
Direktes Bildschirmprogrammieren 85
Festplatte 178
Aufteilung (siehe Disketten) 101
Bereichsbelegung 102, 107
Urladereintrag (Master-Boot-Eintrag) 101

Formate 100, 101
Logisch 100, 103
Physikalisch 100
Partitionen 100
Platte 97
Sektoren 100, 104
Spuren 100
Zylinder 100, 107
Festumschaltungen (siehe Tastatur)
Flaggenregister 32
Floppydisk-Controller (siehe FDC)
Funktionen
Traditionell 249
Erweiterte (neue) 277
Zusammenfassung 250

Gerätetreiber, installierbar 98, 282
Globale Zeichen (siehe Dateigruppenzeichen)
Grafikmodi 66
Grafikzeichen (siehe Bildschirm, ASCII) 385

Hardware 16
Hexadezimale Zahlen 368
Adressen, segmentierte 370
Bitmusterdarstellung 369
BASIC 373
Hexzahlenaddition 375
Hexzahlenmultiplikation 376
Umwandlung dezimal/hexadezimal 371

IBM BASIC Compiler (siehe BASIC)
IBM Pascal Version 1.00 (siehe Pascal)
IBMBIO.COM-Datei 44
IBMDOS.COM-Datei 44
ICA 58
Indexregister (siehe Offsetregister) 26, 31
Quellindex (SI)-Register 31
Zielindex (DI)-Register 31
Initialisierung 43
Intel 8088 (siehe 8088)
Intel 80286 (siehe 80286)
Interrupt 23
Adreßinterrupt 47
Allgemeine Erläuterungen 36
BASIC 47

BIOS 146
CPU 47
DOS 47
Hardware 47
Hauptinterrupts 49
Nicht-maskierbarer Interrupt (NMI) 37
Software 47
Interruptvektoren 46, 48, 266, 286
Verändern/Setzen der Werte mit DOS 48, 50
Vektortabelle 37
IO.SYS (siehe IBMBIO.COM)
IP - Befehlszeiger (siehe Offsetregister)

Kassettenrekorder
Allgemeine Erläuterungen 197
AT-Routinen 199
Routinen 197
Motor an-/ausschalten 198
Schreiben/Lesen von Daten 199
KEEP 285
Kommunikation, intern 58
Kommunikation, seriell 194
Kompiler (siehe Compiler)
Kontrollinformationsbereich 52
Kopierschutz
Allgemeine Erläuterungen 120
Verwendung von BIOS-Routinen 120, 177
Verwendung der Formatierungsroutine 120

Landesabhängige Informationen 287
Lattice/Microsoft C (siehe C-Programmiersprache)
Laufwerk (siehe Laufwerksparametertabelle)
Laufwerksroutinen 172, 179, 262
Laufwerksparametertabelle
Allgemeine Erläuterungen 180
Anlegen einer neuen Tabelle 180
Interruptvektor 180
Komponenten 181
Laufwerkstatus 173, 178, 257
Lautsprecher (siehe Tonerzeugung)
LIB 326
Lichtgriffel 160
LINK-Programm 327, 334
.EXE-Dateien binden 324
An eine Bibliothek binden 327
Verknüpfung von Programmen 328

Linken (siehe LINK-Programm)
Long-Sektoren (AT) 178

Makro-Assembler (siehe Assembler)
Maschinen-ID 58
Matrix (siehe Zeichenfeld)
Memory-Mapped-Anzeige 66, 82
Mikroprozessor 2
Monitore 74
    Direktsignalmonitor 91
    Farbqualität 76
    Mischsignalmonitor 91
    RGB-Monitor 91
Monochromadapter
    Allgemeine Erläuterungen 66
    E/A-Ports 86
    Monochrommodus 66, 77, 157
    Speicherbezogene Darstellung von Zeichen 82
MS-DOS.SYS (siehe IBMDOS.COM)
Multitasking
    Allgemeine Erläuterungen 6
    Direkte Bildschirmausgabe in Multitasking-Umgebungen 85

NEC Controller (siehe PD765)
Neue DOS-Funktionen (siehe erweiterte DOS-Funktionen)
Nicht-Maskierbarer Interrupt 37
NMI (siehe Nicht-Maskierbarer Interrupt)

Objektcode
    Allgemeine Erläuterungen 324
    Bibliotheken anlegen 325
Offsetadressen 25
Offsetregister 31
    Basiszeiger (BP) 31
    Befehlszeiger (IP oder PC) 31
    Quellindex (SI) 31
    Stapelzeiger (SP) 31
    Zielindex (DI) 31

Parameterübergabe
    BASIC 341
    C 359
    Pascal 352
Pascal
    Allgemeine Erläuterungen 331, 347
    Assemblerschnittstellen 352
    Datenformate 348

PD765 Diskettenkontroller 11
Peripherie (siehe PPI)
Pixel
    Allgemeine Erläuterungen 66, 68
    Auflösung 70
    BIOS-Routinen 166
    Speicherschema in Grafikmodi 84
Platine 3,4
Ports
    Adressen 37
    Allgemeine Erläuterungen 16, 23, 35
    Benutzung 36
    Geräte-E/A 35
    Parameter 195
    Seriell (RS-232) 194
    Status 197
    INP/OUT 35
POST 43
PPI (siehe 8255)
Programmlänge 30
Programmoverlay 266
Programmodule 323
Programmschnittstelle (siehe Schnittstellenroutinen)
Programmsegmentpräfix (siehe PSP)
Programmbeendung (siehe Beenden)
Programmierbares Peripherie-Interface (siehe 8255)
Programmierbarer Zeitgeber (siehe 8253)
Programmiersprachen 20, 320, 329
PSP 239, 266, 307
    Adreßinterrupts setzen 234
    CS-Register 239
    DOS-Routinen 266, 307
    Position 239
    Programmbeendung 240
    Struktur 240
    Verwendung 239
    Zugriff über Segmentregister 240
Prüfsummenfehler (CRC) 175, 199, 270

Quellcode 323
Quellindex (SI) (siehe Offsetregister)

Rasterscan 66
Reboot-System 127, 204
Register 26, 32, 34
Relativer Offset (siehe Speicheradressierung)

ROM
Allgemeine Erläuterungen 42
BASIC 44, 61
Kassetten 62
Erweiterungen 43, 62
Freigabedaten 59, 60
Komponenten 42
Modellidentifikation 59
Startprogramme 43
Bootstrap-Loader (Urlader) 44
Initialisierung 43
Power on self test (POST) 43
Verfügbarkeitstest 34
Versionen 58
ROM-BIOS 42, 45, 154
RS-232 194

Scancodes (siehe Tastaturauswahlcode)
Schnittstellenroutinen
Allgemeine Erläuterungen 149
Allgemeine Form 150
Erfordernisse 320
Schnittstellentreiber (siehe Gerätetreiber)
Segmentregister (siehe CS-, DS- ES- UND SS-Register) 26, 29
Segmentierte Notation (siehe Speicheradressierung)
Sektoren (siehe Disketten) 173, 175
Serielle Kommunikation 194
Sharingcodes (siehe Dateisharing)
SI (Quellindex) (siehe Offsetregister)
SP-Register (siehe Offsetregister)
Speicher 14, 23, 298
adressierbarer Speicher 25
64-Kbyte-Blöcke 14
0-Block bis 9-Block 14
A-Block 15
B-Block 15
C-Block 15
D-Block 16
E-Block 16
F-Block 16
20-bit-Adressen 14
Unterer Speicherbereich 51
Speicheradressierung 25, 32
Segmentadressen 25
Notation 25

Speicherresidente Programme 232
Speicherroutinen (siehe BIOS, DOS)
Speicherzugriff (siehe DMA)
Spur (siehe Disketten) 176
    Adressierung 177
    Anzahl 103
    Indexloch 177
    Formatierung 176
SS - Stapelsegmentregister 29
Stapel
    Abbau 322
    Allgemeine Erläuterungen 38, 153
    Auswirkung der EXEC-Routine 300
    BASIC 342, 345
    Größe 39
    Inhalt nach einem kritischen Fehler 237
    LIFO 38
    Parameter
    Zugriffsadresse 153
    Zugriff auf Daten 153
Stapelzeiger (SP) 39
Stapelsegmentregister (siehe SS-Register)
Statuscodes (siehe Fehlercodes)
Stepperzeit (siehe Laufwerksparametertabelle)
String 168, 255
Suche (siehe Dateisuche)
Systemplatine 3, 4, 11, 13
Systemstart 42

Tageszeit (siehe Zeit- und Datumsfunktionen)
Taktgenerator (siehe 8284A)
Tastatur 124
    ANSI-Kontrolle 365
    ASCII-Codes 129
    AT 135
    BIOS-Routinen 188
    CHR$(0) 129
    Cursortasten 190
    Direkte Eingabe 129
    Doppelt vorhandene Tasten 128
    DOS-Routinen 254
    Eingabe 251, 256
    Funktionstasten (siehe Sondertasten)
    Hauptbyte 130, 188
    Hilfsbyte 130, 188

INKEY$ 129
Interrupts 134
Numerische Tasten 130
Puffer 188, 260
Sondertasten 130, 131
Statusbytes 132, 256
Caps Lock 133
Einfügemodus 133
Festumschaltungstasten 134
Haltemodus 133
Tastaturinterrupts (siehe auch BIOS)
Tastaturlayout 125
Tastaturauswahlcodes (Scancodes)
Allgemeine Erläuterungen 125
Speicherung 126
Standardcodes 125
Umsetzung 126
Tastenanschlagpufferung 126
Umschaltstatus 127
Alt-Taste 126
Ctrl-Alt-Del 127
Ctrl-Break 127, 285
Ctrl-Num-Lock 127
Shift-PrtSc 127
Umschalttasten 126
Unterbrechungstaste 127, 235, 252
Wiederholfunktion 128
Textdateien
End-of-file-Marke 387
Formatkonventionen 387
Textmodi 66
Timer (siehe 8253)
Tonerzeugung
BASIC 140
Computererzeugte Töne 139
Frequenzbereich in BASIC 141
Frequenzen von Noten 139
Qualität 144
Lautsprecher aktivieren 142
Lautsprecherkontrolle 141
Direkt 143
Lautstärke 144
Physikalische Grundlagen 138
Zeitgeber (8253) 139, 140, 141
Traditionelle DOS-Funktionen 247
TTY (Teletypemodus) 167

Uhr
BASIC - Takt 58
Mitternachtssignal 56
Systemtakt 56
Zählerstand 56, 205
Umgekehrtes Speichern (von Worten) 39
Umrandungszeichen 384
Unterbrechungstaste (Break) (siehe Tastatur)
Unterverzeichnis 114, 290
Baumstruktur 114
Dateiattribute 114, 294
Länge 114
Stammverzeichnis 114
Urladereintrag (Boot) (siehe ROM)

Vektortabelle (siehe Interruptvektoren)
Vererbungscodes (siehe Dateisharing)
Vertikaler Rücklauf 69
Vertikales Synchronisationssignal 69, 86
Verzeichnis (siehe Dateiverzeichnis) 109, 290, 297
Attribute 111
Dateiname 110, 267
Dateinamenserweiterung 111
Dateilänge 114
Datum 113
Start-Clusternummer 113
Zeit 113
Virtueller Speicher 6

Wiederholfunktion 128
Wort (siehe umgekehrtes Speichern)

Zeichen (siehe ASCII) 24, 384
Zeichenfeld 381
Zeichenformat 381
Zeichenmatrix (siehe Zeichenfeld)
Zeichensatz 378, 380
Zeit- und Datumsfunktionen 269, 302
Zeitgeber (siehe 8253)
Zielindex (DI) (siehe Offsetregister)
Zwischenspeicherregister (siehe Arbeitsregister) 26, 27
Zylinder (siehe Festplatte) 179

6845 CRT Controller 66
8048 Tastatur-Controller 125
8086 Mikroprozessor 5
8087 Arithmetik-Coprozessor 7
8088 Mikroprozessor 2, 21, 23
8089 E/A-Coprozessor 7
8237A DMA Controller 9
8253 Programmierbarer Zeitgeber 10
8253-5 (siehe 8253)
8255 Programmierbares Peripherie-Interface 10
8255A-5 (siehe 8255)
8259 Interrupt-Controller 9
    Verbinden (im AT) 9
8284A Taktgenerator 10
80286 Mikroprozessor 5, 7
80287 Arithmetik-Coprozessor 7

VIEWEG

Ray Duncan

MS-DOS für Fortgeschrittene

Das Microsoft-Handbuch zum Programmieren mit Assembler und C. (Advanced MS-DOS, dt.) Aus dem Amerik. übers. und bearb. von Andreas Dripke und Angelika Schätzel. Ein Microsoft Press/Vieweg-Buch. 1987. X, 473 S. 18,5 x 23,5 cm. Kart.

Inhalt: Die Entwicklung von MS-DOS – Die Arbeitsweise von MS-DOS – Programmieren unter MS-DOS – Einsatz der Programmierhilfen unter MS-DOS – Programmierung zeichenorientierter Ein- und Ausgabegeräte – Manipulation von Dateien und Datensätzen unter MS-DOS – Dateiverzeichnisse, Unterverzeichnisse und Datenträgerkennsatz – Disketten und Platten – Speicherverwaltung – Die EXEC-Funktion – Interruptbearbeitungsroutinen – Installierbare Schnittstellentreiber – Entwicklung von Filtern unter MS-DOS – MS-DOS Programming Reference – IBM PC BIOS Reference – Lotus/Intel/Microsoft Expanded Memory Specification Reference – Index Deutsch/Englisch und Englisch/Deutsch.

Das MS-DOS-Buch für den erfahrenen Programmierer beschreibt neben nützlichen Systemroutinen vor allem die Schnittstelle des Betriebssystems zur Programmiersprache C und Assembler. Das Buch ist ein Kompendium für den anspruchsvollen Systementwickler.

Im Anhang ist eine vollständige Auflistung aller Systemaufrufe und der Gerätetreiber enthalten, die zur professionellen Systemprogrammierung mit MS-DOS (bis Version 3.1) benötigt werden.

Das Microsoft Handbuch ist das authentische Nachschlagewerk für den PC-Programmierer.

Die Software zum Buch:
5 1/4"-Diskette für IBM PC und Kompatible unter MS-DOS.